
Description of STM32L1xx HAL drivers

Introduction

STMCube™ is an STMicroelectronics original initiative to ease developers life by reducing development efforts, time and cost. STM32Cube covers STM32 portfolio.

STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows generating C initialization code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeL1 for STM32L1 series)
 - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
 - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
 - All embedded software utilities coming with a full set of examples.

The HAL drivers layer provides a generic multi instance simple set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the built-upon layers, such as the middleware layer, to implement their functions without knowing in-depth how to use the MCU. This structure improves the library code reusability and guarantees an easy portability on other devices.

The HAL drivers include a complete set of ready-to-use APIs which simplify the user application implementation. As an example, the communication peripherals contain APIs to initialize and configure the peripheral, to manage data transfers based on polling, to handle interrupts or DMA, and to manage communication errors.

The HAL drivers APIs are split into two categories: generic APIs which provide common and generic functions for all the STM32 series and extension APIs which include specific and customized functions for a given family or part number.

The HAL drivers are feature-oriented instead of IP-oriented. As an example, the timer APIs are split into several categories following the functions offered by the IP: basic timer, capture, pulse width modulation (PWM), etc..

The drivers source code is developed in Strict ANSI-C which makes it independent from the development tools. It is checked with CodeSonar™ static analysis tool. It is fully documented and is MISRA-C 2004 compliant.

The HAL drivers layer implements run-time failure detection by checking the input values of all functions. Such dynamic checking contributes to enhance the firmware robustness. Run-time detection is also suitable for user application development and debugging.

This user manual is structured as follows:

- Overview of the HAL drivers
- Detailed description of each peripheral driver: configuration structures, functions, and how to use the given API to build your application.



Contents

1	Acronyms and definitions.....	37
2	Overview of HAL drivers.....	39
2.1	HAL and user-application files.....	39
2.1.1	HAL driver files	39
2.1.2	User-application files	40
2.2	HAL data structures	42
2.2.1	Peripheral handle structures	42
2.2.2	Initialization and configuration structure	44
2.2.3	Specific process structures	44
2.3	API classification	44
2.4	Devices supported by HAL drivers	45
2.5	HAL drivers rules.....	50
2.5.1	HAL API naming rules	50
2.5.2	HAL general naming rules.....	51
2.5.3	HAL interrupt handler and callback functions.....	52
2.6	HAL generic APIs.....	53
2.7	HAL extension APIs	54
2.7.1	HAL extension model overview	54
2.7.2	HAL extension model cases	54
2.8	File inclusion model.....	57
2.9	HAL common resources.....	58
2.10	HAL configuration.....	59
2.11	HAL system peripheral handling	60
2.11.1	Clock.....	60
2.11.2	GPIOs.....	60
2.11.3	Cortex NVIC and SysTick timer.....	62
2.11.4	PWR	63
2.11.5	EXTI.....	63
2.11.6	DMA.....	64
2.12	How to use HAL drivers	65
2.12.1	HAL usage models	65
2.12.2	HAL initialization	66
2.12.3	HAL IO operation process	68
2.12.4	Timeout and error management.....	71
3	HAL System Driver	76

3.1	HAL System Driver.....	76
3.2	HAL Firmware driver API description	76
3.2.1	How to use this driver	76
3.2.2	Initialization and de-initialization functions	76
3.2.3	HAL Control functions.....	76
3.2.4	HAL_Init.....	77
3.2.5	HAL_DeInit	77
3.2.6	HAL_MspInit	77
3.2.7	HAL_MspDeInit	78
3.2.8	HAL_InitTick	78
3.2.9	HAL_IncTick	78
3.2.10	HAL_GetTick	78
3.2.11	HAL_Delay	78
3.2.12	HAL_SuspendTick.....	79
3.2.13	HAL_ResumeTick.....	79
3.2.14	HAL_GetHalVersion	79
3.2.15	HAL_GetREVID.....	79
3.2.16	HAL_GetDEVID.....	80
3.2.17	HAL_DBGMCU_EnableDBGSleepMode	80
3.2.18	HAL_DBGMCU_DisableDBGSleepMode	80
3.2.19	HAL_DBGMCU_EnableDBGStopMode	80
3.2.20	HAL_DBGMCU_DisableDBGStopMode	80
3.2.21	HAL_DBGMCU_EnableDBGStandbyMode	80
3.2.22	HAL_DBGMCU_DisableDBGStandbyMode	80
3.3	HAL Firmware driver defines.....	80
3.3.1	HAL.....	80
4	HAL ADC Generic Driver.....	82
4.1	HAL ADC Generic Driver	82
4.2	ADC Firmware driver registers structures	82
4.2.1	ADC_InitTypeDef.....	82
4.2.2	ADC_ChannelConfTypeDef	84
4.2.3	ADC_AnalogWDGConfTypeDef.....	85
4.2.4	ADC_HandleTypeDef	86
4.3	ADC Firmware driver API description.....	86
4.3.1	ADC peripheral features	86
4.3.2	How to use this driver	87
4.3.3	Initialization and de-initialization functions	90

4.3.4	IO operation functions	90
4.3.5	Peripheral Control functions	90
4.3.6	Peripheral State and Errors functions	91
4.3.7	HAL_ADC_Init	91
4.3.8	HAL_ADC_DeInit	91
4.3.9	HAL_ADC_MspInit	92
4.3.10	HAL_ADC_MspDeInit	92
4.3.11	HAL_ADC_Start	92
4.3.12	HAL_ADC_Stop	92
4.3.13	HAL_ADC_PollForConversion	92
4.3.14	HAL_ADC_PollForEvent	93
4.3.15	HAL_ADC_Start_IT	93
4.3.16	HAL_ADC_Stop_IT	93
4.3.17	HAL_ADC_Start_DMA	93
4.3.18	HAL_ADC_Stop_DMA	94
4.3.19	HAL_ADC_GetValue	94
4.3.20	HAL_ADC_IRQHandler	94
4.3.21	HAL_ADC_ConvCpltCallback	94
4.3.22	HAL_ADC_ConvHalfCpltCallback	94
4.3.23	HAL_ADC_LevelOutOfWindowCallback	95
4.3.24	HAL_ADC_ErrorCallback	95
4.3.25	HAL_ADC_ConfigChannel	95
4.3.26	HAL_ADC_AnalogWDGConfig	95
4.3.27	HAL_ADC_GetState	96
4.3.28	HAL_ADC_GetError	96
4.4	ADC Firmware driver defines	96
4.4.1	ADC	96
5	HAL ADC Extension Driver	111
5.1	HAL ADC Extension Driver	111
5.2	ADCEX Firmware driver registers structures	111
5.2.1	ADC_InjectionConfTypeDef	111
5.3	ADCEX Firmware driver API description	113
5.3.1	IO operation functions	113
5.3.2	Peripheral Control functions	113
5.3.3	HAL_ADCEX_InjectedStart	113
5.3.4	HAL_ADCEX_InjectedStop	113
5.3.5	HAL_ADCEX_InjectedPollForConversion	114
5.3.6	HAL_ADCEX_InjectedStart_IT	114

5.3.7	HAL_ADCEx_InjectedStop_IT	114
5.3.8	HAL_ADCEx_InjectedGetValue	114
5.3.9	HAL_ADCEx_InjectedConvCpltCallback	114
5.3.10	HAL_ADCEx_InjectedConfigChannel	115
5.4	ADCEx Firmware driver defines	115
5.4.1	ADCEx	115
6	HAL COMP Generic Driver	120
6.1	HAL COMP Generic Driver	120
6.2	COMP Firmware driver registers structures	120
6.2.1	COMP_InitTypeDef	120
6.2.2	COMP_HandleTypeDef	121
6.3	COMP Firmware driver API description	121
6.3.1	COMP Peripheral features	121
6.3.2	How to use this driver	122
6.3.3	Initialization and de-initialization functions	123
6.3.4	IO operation functions	123
6.3.5	Peripheral Control functions	123
6.3.6	Peripheral State functions	124
6.3.7	HAL_COMP_Init	124
6.3.8	HAL_COMP_DeInit	124
6.3.9	HAL_COMP_MspInit	124
6.3.10	HAL_COMP_MspDeInit	124
6.3.11	HAL_COMP_Start	124
6.3.12	HAL_COMP_Stop	125
6.3.13	HAL_COMP_Start_IT	125
6.3.14	HAL_COMP_Stop_IT	125
6.3.15	HAL_COMP_IRQHandler	125
6.3.16	HAL_COMP_Lock	125
6.3.17	HAL_COMP_GetOutputLevel	126
6.3.18	HAL_COMP_TriggerCallback	126
6.3.19	HAL_COMP_GetState	126
6.4	COMP Firmware driver defines	126
6.4.1	COMP	126
7	HAL COMP Extension Driver	134
7.1	HAL COMP Extension Driver	134
7.2	COMPEx Firmware driver defines	134
7.2.1	COMPEx	134

8	HAL CORTEX Generic Driver.....	137
8.1	HAL CORTEX Generic Driver	137
8.2	CORTEX Firmware driver registers structures	137
8.2.1	MPU_Region_InitTypeDef.....	137
8.3	CORTEX Firmware driver API description	138
8.3.1	Initialization and de-initialization functions	138
8.3.2	Peripheral Control functions	138
8.3.3	HAL_NVIC_SetPriorityGrouping	138
8.3.4	HAL_NVIC_SetPriority	139
8.3.5	HAL_NVIC_EnableIRQ	139
8.3.6	HAL_NVIC_DisableIRQ.....	139
8.3.7	HAL_NVIC_SystemReset.....	139
8.3.8	HAL_SYSTICK_Config.....	140
8.3.9	HAL_MPU_ConfigRegion.....	140
8.3.10	HAL_NVIC_GetPriorityGrouping	140
8.3.11	HAL_NVIC_GetPriority	140
8.3.12	HAL_NVIC_SetPendingIRQ	141
8.3.13	HAL_NVIC_GetPendingIRQ	141
8.3.14	HAL_NVIC_ClearPendingIRQ.....	141
8.3.15	HAL_NVIC_GetActive	141
8.3.16	HAL_SYSTICK_CLKSourceConfig	142
8.3.17	HAL_SYSTICK_IRQHandler	142
8.3.18	HAL_SYSTICK_Callback	142
8.4	CORTEX Firmware driver defines.....	142
8.4.1	CORTEX.....	142
9	HAL CRC Generic Driver.....	146
9.1	HAL CRC Generic Driver	146
9.2	CRC Firmware driver registers structures	146
9.2.1	CRC_HandleTypeDef.....	146
9.3	CRC Firmware driver API description	146
9.3.1	How to use this driver	146
9.3.2	Initialization and de-initialization functions	146
9.3.3	Peripheral Control functions	147
9.3.4	Peripheral State functions	147
9.3.5	HAL_CRC_Init.....	147
9.3.6	HAL_CRC_DeInit	147
9.3.7	HAL_CRC_MspInit	147

9.3.8	HAL_CRC_MspDeInit.....	148
9.3.9	HAL_CRC_Accumulate.....	148
9.3.10	HAL_CRC_Calculate.....	148
9.3.11	HAL_CRC_GetState.....	148
9.3.12	HAL_CRC_Accumulate.....	148
9.3.13	HAL_CRC_Calculate.....	149
9.4	CRC Firmware driver defines.....	149
9.4.1	CRC.....	149
10	HAL CRYPT Generic Driver.....	151
10.1	HAL CRYPT Generic Driver.....	151
10.2	CRYPT Firmware driver registers structures.....	151
10.2.1	CRYPT_InitTypeDef.....	151
10.2.2	CRYPT_HandleTypeDef.....	151
10.3	CRYPT Firmware driver API description.....	152
10.3.1	Initialization and de-initialization functions.....	152
10.3.2	AES processing functions.....	152
10.3.3	DMA callback functions.....	153
10.3.4	CRYPT IRQ handler management.....	153
10.3.5	Peripheral State functions.....	153
10.3.6	HAL_CRYPT_Init.....	153
10.3.7	HAL_CRYPT_DeInit.....	154
10.3.8	HAL_CRYPT_MspInit.....	154
10.3.9	HAL_CRYPT_MspDeInit.....	154
10.3.10	HAL_CRYPT_AESECB_Encrypt.....	154
10.3.11	HAL_CRYPT_AESCBC_Encrypt.....	154
10.3.12	HAL_CRYPT_AESCTR_Encrypt.....	155
10.3.13	HAL_CRYPT_AESECB_Decrypt.....	155
10.3.14	HAL_CRYPT_AESCBC_Decrypt.....	155
10.3.15	HAL_CRYPT_AESCTR_Decrypt.....	156
10.3.16	HAL_CRYPT_AESECB_Encrypt_IT.....	156
10.3.17	HAL_CRYPT_AESCBC_Encrypt_IT.....	156
10.3.18	HAL_CRYPT_AESCTR_Encrypt_IT.....	157
10.3.19	HAL_CRYPT_AESECB_Decrypt_IT.....	157
10.3.20	HAL_CRYPT_AESCBC_Decrypt_IT.....	157
10.3.21	HAL_CRYPT_AESCTR_Decrypt_IT.....	158
10.3.22	HAL_CRYPT_AESECB_Encrypt_DMA.....	158
10.3.23	HAL_CRYPT_AESCBC_Encrypt_DMA.....	158
10.3.24	HAL_CRYPT_AESCTR_Encrypt_DMA.....	159

10.3.25	HAL_CRYP_AESECB_Decrypt_DMA	159
10.3.26	HAL_CRYP_AESCBC_Decrypt_DMA	159
10.3.27	HAL_CRYP_AESCTR_Decrypt_DMA	160
10.3.28	HAL_CRYP_ErrorCallback.....	160
10.3.29	HAL_CRYP_InCpltCallback	160
10.3.30	HAL_CRYP_OutCpltCallback	160
10.3.31	HAL_CRYP_IRQHandler.....	160
10.3.32	HAL_CRYP_GetState	161
10.4	CRYP Firmware driver defines.....	161
10.4.1	CRYP.....	161
11	HAL CRYP Extension Driver	165
11.1	HAL CRYP Extension Driver	165
11.2	CRYPEx Firmware driver API description	165
11.2.1	Extended features functions	165
11.2.2	HAL_CRYPEx_ComputationCpltCallback.....	165
11.3	CRYPEx Firmware driver defines.....	165
11.3.1	CRYPEx	165
12	HAL DAC Generic Driver.....	166
12.1	HAL DAC Generic Driver	166
12.2	DAC Firmware driver registers structures	166
12.2.1	DAC_HandleTypeDef	166
12.2.2	DAC_ChannelConfTypeDef	166
12.3	DAC Firmware driver API description.....	167
12.3.1	DAC Peripheral features.....	167
12.3.2	How to use this driver	168
12.3.3	Initialization and de-initialization functions	169
12.3.4	IO operation functions	169
12.3.5	Peripheral Control functions	170
12.3.6	Peripheral State and Errors functions	170
12.3.7	HAL_DAC_Init	170
12.3.8	HAL_DAC_DeInit.....	170
12.3.9	HAL_DAC_MspInit	171
12.3.10	HAL_DAC_MspDeInit.....	171
12.3.11	HAL_DAC_Start	171
12.3.12	HAL_DAC_Stop.....	171
12.3.13	HAL_DAC_Start_DMA	172
12.3.14	HAL_DAC_Stop_DMA.....	172

12.3.15	HAL_DAC_GetValue	172
12.3.16	HAL_DAC_IRQHandler	172
12.3.17	HAL_DAC_ConvCpltCallbackCh1	173
12.3.18	HAL_DAC_ConvHalfCpltCallbackCh1	173
12.3.19	HAL_DAC_ErrorCallbackCh1	173
12.3.20	HAL_DAC_DMAUnderrunCallbackCh1	173
12.3.21	HAL_DAC_SetValue	173
12.3.22	HAL_DAC_ConfigChannel	174
12.3.23	HAL_DAC_GetState	174
12.3.24	HAL_DAC_GetError	174
12.3.25	HAL_DAC_ConfigChannel	175
12.3.26	HAL_DAC_SetValue	175
12.3.27	HAL_DAC_GetState	175
12.3.28	HAL_DAC_GetError	175
12.4	DAC Firmware driver defines	176
12.4.1	DAC	176
13	HAL DAC Extension Driver	180
13.1	HAL DAC Extension Driver	180
13.2	DACEx Firmware driver API description	180
13.2.1	How to use this driver	180
13.2.2	Extended features functions	180
13.2.3	HAL_DACEx_DualGetValue	180
13.2.4	HAL_DACEx_TriangleWaveGenerate	180
13.2.5	HAL_DACEx_NoiseWaveGenerate	181
13.2.6	HAL_DACEx_DualSetValue	182
13.2.7	HAL_DACEx_ConvCpltCallbackCh2	182
13.2.8	HAL_DACEx_ConvHalfCpltCallbackCh2	182
13.2.9	HAL_DACEx_ErrorCallbackCh2	183
13.2.10	HAL_DACEx_DMAUnderrunCallbackCh2	183
13.3	DACEx Firmware driver defines	183
13.3.1	DACEx	183
14	HAL DMA Generic Driver	185
14.1	HAL DMA Generic Driver	185
14.2	DMA Firmware driver registers structures	185
14.2.1	DMA_InitTypeDef	185
14.2.2	__DMA_HandleTypeDef	185
14.3	DMA Firmware driver API description	186

14.3.1	How to use this driver	186
14.3.2	Initialization and de-initialization functions	187
14.3.3	IO operation functions	187
14.3.4	State and Errors functions	188
14.3.5	HAL_DMA_Init	188
14.3.6	HAL_DMA_DeInit	188
14.3.7	HAL_DMA_Start	188
14.3.8	HAL_DMA_Start_IT	189
14.3.9	HAL_DMA_Abort	189
14.3.10	HAL_DMA_PollForTransfer	189
14.3.11	HAL_DMA_IRQHandler	190
14.3.12	HAL_DMA_GetState	190
14.3.13	HAL_DMA_GetError	190
14.4	DMA Firmware driver defines	190
14.4.1	DMA	190
15	HAL DMA Extension Driver	195
15.1	HAL DMA Extension Driver	195
15.2	DMAEx Firmware driver defines	195
15.2.1	DMAEx	195
16	HAL FLASH Generic Driver	197
16.1	HAL FLASH Generic Driver	197
16.2	FLASH Firmware driver registers structures	197
16.2.1	FLASH_ProcessTypeDef	197
16.3	FLASH Firmware driver API description	197
16.3.1	FLASH peripheral features	197
16.3.2	How to use this driver	197
16.3.3	Programming operation functions	198
16.3.4	Option Bytes Programming functions	199
16.3.5	Peripheral Control functions	199
16.3.6	Peripheral Errors functions	199
16.3.7	HAL_FLASH_Program	200
16.3.8	HAL_FLASH_Program_IT	200
16.3.9	HAL_FLASH_EndOfOperationCallback	200
16.3.10	HAL_FLASH_OperationErrorCallback	200
16.3.11	HAL_FLASH_IRQHandler	201
16.3.12	HAL_FLASH_Unlock	201
16.3.13	HAL_FLASH_Lock	201

16.3.14	HAL_FLASH_OB_Unlock.....	201
16.3.15	HAL_FLASH_OB_Lock.....	201
16.3.16	HAL_FLASH_OB_Launch.....	201
16.3.17	HAL_FLASH_GetError.....	201
16.4	FLASH Firmware driver defines	202
16.4.1	FLASH.....	202
17	HAL FLASH Extension Driver	206
17.1	HAL FLASH Extension Driver	206
17.2	FLASHEx Firmware driver registers structures	206
17.2.1	FLASH_EraseInitTypeDef	206
17.2.2	FLASH_OBProgramInitTypeDef	206
17.2.3	FLASH_AdvOBProgramInitTypeDef	207
17.3	FLASHEx Firmware driver API description.....	207
17.3.1	FLASH Erasing Programming functions.....	207
17.3.2	Option Bytes Programming functions.....	208
17.3.3	DATA EEPROM Programming functions	208
17.3.4	HAL_FLASHEx_Erase	209
17.3.5	HAL_FLASHEx_Erase_IT	209
17.3.6	HAL_FLASHEx_OBProgram.....	209
17.3.7	HAL_FLASHEx_OBGetConfig	210
17.3.8	HAL_FLASHEx_AdvOBProgram	210
17.3.9	HAL_FLASHEx_AdvOBGetConfig	210
17.3.10	HAL_FLASHEx_DATAEEPROM_Unlock	210
17.3.11	HAL_FLASHEx_DATAEEPROM_Lock.....	210
17.3.12	HAL_FLASHEx_DATAEEPROM_Erase	211
17.3.13	HAL_FLASHEx_DATAEEPROM_Program.....	211
17.3.14	HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram	211
17.3.15	HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram.....	212
17.4	FLASHEx Firmware driver defines	212
17.4.1	FLASHEx.....	212
18	HAL FLASH__RAMFUNC Generic Driver	219
18.1	HAL FLASH__RAMFUNC Generic Driver	219
18.2	FLASH__RAMFUNC Firmware driver API description.....	219
18.2.1	HAL_FLASHEx_EnableRunPowerDown	219
18.2.2	HAL_FLASHEx_DisableRunPowerDown.....	219
18.2.3	HAL_FLASHEx_EraseParallelPage.....	219
18.2.4	HAL_FLASHEx_ProgramParallelHalfPage.....	219

18.2.5	HAL_FLASHEx_HalfPageProgram	220
18.2.6	HAL_FLASHEx_DATAEEPROM_EraseDoubleWord	221
18.2.7	HAL_FLASHEx_DATAEEPROM_ProgramDoubleWord	221
19	HAL GPIO Generic Driver.....	223
19.1	HAL GPIO Generic Driver	223
19.2	GPIO Firmware driver registers structures	223
19.2.1	GPIO_InitTypeDef	223
19.3	GPIO Firmware driver API description	223
19.3.1	GPIO Peripheral features	223
19.3.2	How to use this driver	224
19.3.3	Initialization and Configuration functions	225
19.3.4	HAL_GPIO_Init	225
19.3.5	HAL_GPIO_DeInit	225
19.3.6	HAL_GPIO_ReadPin	225
19.3.7	HAL_GPIO_WritePin	225
19.3.8	HAL_GPIO_TogglePin	226
19.3.9	HAL_GPIO_LockPin	226
19.3.10	HAL_GPIO_EXTI_IRQHandler	227
19.3.11	HAL_GPIO_EXTI_Callback	227
19.4	GPIO Firmware driver defines	227
19.4.1	GPIO	227
20	HAL GPIO Extension Driver	231
20.1	HAL GPIO Extension Driver	231
20.2	GPIOEx Firmware driver defines	231
20.2.1	GPIOEx	231
21	HAL I2C Generic Driver	233
21.1	HAL I2C Generic Driver	233
21.2	I2C Firmware driver registers structures	233
21.2.1	I2C_InitTypeDef	233
21.2.2	I2C_HandleTypeDef	233
21.3	I2C Firmware driver API description	234
21.3.1	How to use this driver	234
21.3.2	Initialization and de-initialization functions	237
21.3.3	IO operation functions	237
21.3.4	Peripheral State and Errors functions	239
21.3.5	HAL_I2C_Init	239
21.3.6	HAL_I2C_DeInit	239

21.3.7	HAL_I2C_MspInit	239
21.3.8	HAL_I2C_MspDeInit.....	239
21.3.9	HAL_I2C_Master_Transmit.....	240
21.3.10	HAL_I2C_Master_Receive	240
21.3.11	HAL_I2C_Slave_Transmit.....	240
21.3.12	HAL_I2C_Slave_Receive	241
21.3.13	HAL_I2C_Master_Transmit_IT.....	241
21.3.14	HAL_I2C_Master_Receive_IT.....	241
21.3.15	HAL_I2C_Slave_Transmit_IT.....	241
21.3.16	HAL_I2C_Slave_Receive_IT.....	242
21.3.17	HAL_I2C_Master_Transmit_DMA.....	242
21.3.18	HAL_I2C_Master_Receive_DMA.....	242
21.3.19	HAL_I2C_Slave_Transmit_DMA.....	242
21.3.20	HAL_I2C_Slave_Receive_DMA.....	243
21.3.21	HAL_I2C_Mem_Write.....	243
21.3.22	HAL_I2C_Mem_Read	243
21.3.23	HAL_I2C_Mem_Write_IT	244
21.3.24	HAL_I2C_Mem_Read_IT	244
21.3.25	HAL_I2C_Mem_Write_DMA	244
21.3.26	HAL_I2C_Mem_Read_DMA	245
21.3.27	HAL_I2C_IsDeviceReady.....	245
21.3.28	HAL_I2C_EV_IRQHandler	245
21.3.29	HAL_I2C_ER_IRQHandler	246
21.3.30	HAL_I2C_MasterTxCpltCallback.....	246
21.3.31	HAL_I2C_MasterRxCpltCallback	246
21.3.32	HAL_I2C_SlaveTxCpltCallback.....	246
21.3.33	HAL_I2C_SlaveRxCpltCallback	246
21.3.34	HAL_I2C_MemTxCpltCallback.....	246
21.3.35	HAL_I2C_MemRxCpltCallback	247
21.3.36	HAL_I2C_ErrorCallback	247
21.3.37	HAL_I2C_GetState	247
21.3.38	HAL_I2C_GetError	247
21.4	I2C Firmware driver defines	247
21.4.1	I2C	247
22	HAL I2S Generic Driver	254
22.1	HAL I2S Generic Driver.....	254
22.2	I2S Firmware driver registers structures	254
22.2.1	I2S_InitTypeDef.....	254

22.2.2	I2S_HandleTypeDef	254
22.3	I2S Firmware driver API description	255
22.3.1	How to use this driver	255
22.3.2	Initialization and de-initialization functions	257
22.3.3	IO operation functions	257
22.3.4	Peripheral State and Errors functions	258
22.3.5	HAL_I2S_Init	258
22.3.6	HAL_I2S_DeInit	258
22.3.7	HAL_I2S_MspInit	259
22.3.8	HAL_I2S_MspDeInit	259
22.3.9	HAL_I2S_Transmit	259
22.3.10	HAL_I2S_Receive	259
22.3.11	HAL_I2S_Transmit_IT	260
22.3.12	HAL_I2S_Receive_IT	260
22.3.13	HAL_I2S_Transmit_DMA	261
22.3.14	HAL_I2S_Receive_DMA	261
22.3.15	HAL_I2S_DMAPause	261
22.3.16	HAL_I2S_DMAResume	262
22.3.17	HAL_I2S_DMAStop	262
22.3.18	HAL_I2S_IRQHandler	262
22.3.19	HAL_I2S_TxHalfCpltCallback	262
22.3.20	HAL_I2S_TxCpltCallback	262
22.3.21	HAL_I2S_RxHalfCpltCallback	263
22.3.22	HAL_I2S_RxCpltCallback	263
22.3.23	HAL_I2S_ErrorCallback	263
22.3.24	HAL_I2S_GetState	263
22.3.25	HAL_I2S_GetError	263
22.4	I2S Firmware driver defines	264
22.4.1	I2S	264
23	HAL IRDA Generic Driver	268
23.1	HAL IRDA Generic Driver	268
23.2	IRDA Firmware driver registers structures	268
23.2.1	IRDA_InitTypeDef	268
23.2.2	IRDA_HandleTypeDef	268
23.3	IRDA Firmware driver API description	269
23.3.1	How to use this driver	269
23.3.2	Initialization and Configuration functions	271
23.3.3	IO operation functions	272

23.3.4	Peripheral State and Errors functions	273
23.3.5	HAL_IRDA_Init	273
23.3.6	HAL_IRDA_DeInit.....	273
23.3.7	HAL_IRDA_MspInit	273
23.3.8	HAL_IRDA_MspDeInit.....	274
23.3.9	HAL_IRDA_Transmit.....	274
23.3.10	HAL_IRDA_Receive	274
23.3.11	HAL_IRDA_Transmit_IT	274
23.3.12	HAL_IRDA_Receive_IT	275
23.3.13	HAL_IRDA_Transmit_DMA.....	275
23.3.14	HAL_IRDA_Receive_DMA.....	275
23.3.15	HAL_IRDA_DMABase.....	275
23.3.16	HAL_IRDA_DMAResume.....	276
23.3.17	HAL_IRDA_DMAStop.....	276
23.3.18	HAL_IRDA_IRQHandler	276
23.3.19	HAL_IRDA_TxCpltCallback.....	276
23.3.20	HAL_IRDA_TxHalfCpltCallback	276
23.3.21	HAL_IRDA_RxCpltCallback	277
23.3.22	HAL_IRDA_RxHalfCpltCallback.....	277
23.3.23	HAL_IRDA_ErrorCallback	277
23.3.24	HAL_IRDA_GetState.....	277
23.3.25	HAL_IRDA_GetError	277
23.4	IRDA Firmware driver defines	278
23.4.1	IRDA	278
24	HAL IWDG Generic Driver	286
24.1	HAL IWDG Generic Driver	286
24.2	IWDG Firmware driver registers structures	286
24.2.1	IWDG_InitTypeDef	286
24.2.2	IWDG_HandleTypeDef.....	286
24.3	IWDG Firmware driver API description	286
24.3.1	Initialization and de-initialization functions	286
24.3.2	IO operation functions	287
24.3.3	Peripheral State functions	287
24.3.4	HAL_IWDG_Init	287
24.3.5	HAL_IWDG_MspInit	287
24.3.6	HAL_IWDG_Start	287
24.3.7	HAL_IWDG_Refresh	288
24.3.8	HAL_IWDG_GetState.....	288

24.4	IWDG Firmware driver defines	288
24.4.1	IWDG	288
25	HAL LCD Generic Driver	291
25.1	HAL LCD Generic Driver	291
25.2	LCD Firmware driver registers structures	291
25.2.1	LCD_InitTypeDef	291
25.2.2	LCD_HandleTypeDef	292
25.3	LCD Firmware driver API description	292
25.3.1	How to use this driver	292
25.3.2	Initialization and Configuration functions	293
25.3.3	IO operation functions	293
25.3.4	Peripheral State functions	293
25.3.5	HAL_LCD_DeInit	294
25.3.6	HAL_LCD_Init	294
25.3.7	HAL_LCD_MspDeInit	294
25.3.8	HAL_LCD_MspInit	294
25.3.9	HAL_LCD_Write	294
25.3.10	HAL_LCD_Clear	295
25.3.11	HAL_LCD_UpdateDisplayRequest	295
25.3.12	HAL_LCD_GetState	296
25.3.13	HAL_LCD_GetError	296
25.4	LCD Firmware driver defines	296
25.4.1	LCD	296
26	HAL NOR Generic Driver	306
26.1	HAL NOR Generic Driver	306
26.2	NOR Firmware driver registers structures	306
26.2.1	NOR_IDTypeDef	306
26.2.2	NOR_CFITypeDef	306
26.2.3	NOR_HandleTypeDef	307
26.3	NOR Firmware driver API description	307
26.3.1	How to use this driver	307
26.3.2	NOR Initialization and de_initialization functions	308
26.3.3	NOR Input and Output functions	308
26.3.4	NOR Control functions	308
26.3.5	NOR State functions	308
26.3.6	HAL_NOR_Init	308
26.3.7	HAL_NOR_DeInit	309

26.3.8	HAL_NOR_MspInit	309
26.3.9	HAL_NOR_MspDeInit	309
26.3.10	HAL_NOR_MspWait	309
26.3.11	HAL_NOR_Read_ID	309
26.3.12	HAL_NOR_ReturnToReadMode	310
26.3.13	HAL_NOR_Read	310
26.3.14	HAL_NOR_Program	310
26.3.15	HAL_NOR_ReadBuffer	310
26.3.16	HAL_NOR_ProgramBuffer	311
26.3.17	HAL_NOR_Erase_Block	311
26.3.18	HAL_NOR_Erase_Chip	311
26.3.19	HAL_NOR_Read_CFI	311
26.3.20	HAL_NOR_WriteOperation_Enable	312
26.3.21	HAL_NOR_WriteOperation_Disable	312
26.3.22	HAL_NOR_GetState	312
26.3.23	HAL_NOR_GetStatus	312
26.4	NOR Firmware driver defines	313
26.4.1	NOR	313
27	HAL OPAMP Generic Driver	315
27.1	HAL OPAMP Generic Driver	315
27.2	OPAMP Firmware driver registers structures	315
27.2.1	OPAMP_InitTypeDef	315
27.2.2	OPAMP_HandleTypeDef	316
27.3	OPAMP Firmware driver API description	316
27.3.1	OPAMP Peripheral Features	316
27.3.2	How to use this driver	318
27.3.3	Initialization and de-initialization functions	319
27.3.4	IO operation functions	319
27.3.5	Peripheral Control functions	319
27.3.6	Peripheral State functions	319
27.3.7	HAL_OPAMP_Init	319
27.3.8	HAL_OPAMP_DeInit	320
27.3.9	HAL_OPAMP_MspInit	320
27.3.10	HAL_OPAMP_MspDeInit	320
27.3.11	HAL_OPAMP_Start	320
27.3.12	HAL_OPAMP_Stop	320
27.3.13	HAL_OPAMP_SelfCalibrate	321
27.3.14	HAL_OPAMP_Lock	321

27.3.15	HAL_OPAMP_GetTrimOffset	321
27.3.16	HAL_OPAMP_GetState	321
27.4	OPAMP Firmware driver defines	322
27.4.1	OPAMP	322
28	HAL OPAMP Extension Driver	327
28.1	HAL OPAMP Extension Driver	327
28.2	OPAMPEX Firmware driver API description	327
28.2.1	Peripheral Control functions	327
28.2.2	Extended IO operation functions	327
28.2.3	HAL_OPAMPEX_Unlock	327
28.2.4	HAL_OPAMPEX_SelfCalibrateAll	327
28.3	OPAMPEX Firmware driver defines	328
28.3.1	OPAMPEX	328
29	HAL PCD Generic Driver	330
29.1	HAL PCD Generic Driver	330
29.2	PCD Firmware driver registers structures	330
29.2.1	PCD_InitTypeDef	330
29.2.2	PCD_EPTTypeDef	330
29.2.3	PCD_HandleTypeDef	331
29.3	PCD Firmware driver API description	332
29.3.1	How to use this driver	332
29.3.2	Initialization and de-initialization functions	332
29.3.3	IO operation functions	333
29.3.4	Peripheral Control functions	333
29.3.5	Peripheral State functions	333
29.3.6	HAL_PCD_Init	333
29.3.7	HAL_PCD_DeInit	334
29.3.8	HAL_PCD_MspInit	334
29.3.9	HAL_PCD_MspDeInit	334
29.3.10	HAL_PCD_Start	334
29.3.11	HAL_PCD_Stop	334
29.3.12	HAL_PCD_IRQHandler	335
29.3.13	HAL_PCD_DataOutStageCallback	335
29.3.14	HAL_PCD_DataInStageCallback	335
29.3.15	HAL_PCD_SetupStageCallback	335
29.3.16	HAL_PCD_SOFCallback	335
29.3.17	HAL_PCD_ResetCallback	335

29.3.18	HAL_PCD_SuspendCallback	336
29.3.19	HAL_PCD_ResumeCallback	336
29.3.20	HAL_PCD_ISOOUTIncompleteCallback	336
29.3.21	HAL_PCD_ISOINIncompleteCallback	336
29.3.22	HAL_PCD_ConnectCallback	336
29.3.23	HAL_PCD_DisconnectCallback	336
29.3.24	HAL_PCD_DevConnect	337
29.3.25	HAL_PCD_DevDisconnect	337
29.3.26	HAL_PCD_SetAddress	337
29.3.27	HAL_PCD_EP_Open	337
29.3.28	HAL_PCD_EP_Close	337
29.3.29	HAL_PCD_EP_Receive	338
29.3.30	HAL_PCD_EP_GetRxCount	338
29.3.31	HAL_PCD_EP_Transmit	338
29.3.32	HAL_PCD_EP_SetStall	338
29.3.33	HAL_PCD_EP_ClrStall	339
29.3.34	HAL_PCD_EP_Flush	339
29.3.35	HAL_PCD_ActivateRemoteWakeup	339
29.3.36	HAL_PCD_DeActivateRemoteWakeup	339
29.3.37	HAL_PCD_GetState	339
29.3.38	HAL_PCDEx_SetConnectionState	340
29.4	PCD Firmware driver defines	340
29.4.1	PCD	340
30	HAL PCD Extension Driver	349
30.1	HAL PCD Extension Driver	349
30.2	PCDEx Firmware driver API description	349
30.2.1	Peripheral Control functions	349
30.2.2	HAL_PCDEx_PMAConfig	349
30.3	PCDEx Firmware driver defines	349
30.3.1	PCDEx	349
31	HAL PWR Generic Driver	350
31.1	HAL PWR Generic Driver	350
31.2	PWR Firmware driver registers structures	350
31.2.1	PWR_PVDTypeDef	350
31.3	PWR Firmware driver API description	350
31.3.1	Initialization and de-initialization functions	350
31.3.2	Peripheral Control functions	350

31.3.3	HAL_PWR_DeInit.....	354
31.3.4	HAL_PWR_EnableBkUpAccess	354
31.3.5	HAL_PWR_DisableBkUpAccess.....	354
31.3.6	HAL_PWR_ConfigPVD	355
31.3.7	HAL_PWR_EnablePVD.....	355
31.3.8	HAL_PWR_DisablePVD.....	355
31.3.9	HAL_PWR_EnableWakeUpPin.....	355
31.3.10	HAL_PWR_DisableWakeUpPin	355
31.3.11	HAL_PWR_EnterSLEEPMode.....	356
31.3.12	HAL_PWR_EnterSTOPMode.....	356
31.3.13	HAL_PWR_EnterSTANDBYMode	356
31.3.14	HAL_PWR_EnableSleepOnExit.....	357
31.3.15	HAL_PWR_DisableSleepOnExit	357
31.3.16	HAL_PWR_EnableSEVOnPend	357
31.3.17	HAL_PWR_DisableSEVOnPend.....	357
31.3.18	HAL_PWR_PVD_IRQHandler.....	358
31.3.19	HAL_PWR_PVDCallback	358
31.4	PWR Firmware driver defines	358
31.4.1	PWR	358
32	HAL PWR Extension Driver	364
32.1	HAL PWR Extension Driver.....	364
32.2	PWREx Firmware driver API description.....	364
32.2.1	Peripheral extended features functions.....	364
32.2.2	HAL_PWREx_GetVoltageRange	364
32.2.3	HAL_PWREx_EnableFastWakeUp.....	364
32.2.4	HAL_PWREx_DisableFastWakeUp.....	364
32.2.5	HAL_PWREx_EnableUltraLowPower	364
32.2.6	HAL_PWREx_DisableUltraLowPower	365
32.2.7	HAL_PWREx_EnableLowPowerRunMode	365
32.2.8	HAL_PWREx_DisableLowPowerRunMode	365
32.3	PWREx Firmware driver defines	365
32.3.1	PWREx	365
33	HAL RCC Generic Driver.....	366
33.1	HAL RCC Generic Driver	366
33.2	RCC Firmware driver registers structures	366
33.2.1	RCC_PLLInitTypeDef	366
33.2.2	RCC_OscInitTypeDef	366

33.2.3	RCC_ClkInitTypeDef	367
33.3	RCC Firmware driver API description	368
33.3.1	RCC specific features	368
33.3.2	RCC Limitations	368
33.3.3	Initialization and de-initialization function	368
33.3.4	Peripheral Control functions	370
33.3.5	HAL_RCC_DeInit	370
33.3.6	HAL_RCC_OscConfig	370
33.3.7	HAL_RCC_ClockConfig	370
33.3.8	HAL_RCC_MCOConfig	371
33.3.9	HAL_RCC_EnableCSS	372
33.3.10	HAL_RCC_DisableCSS	372
33.3.11	HAL_RCC_GetSysClockFreq	372
33.3.12	HAL_RCC_GetHCLKFreq	373
33.3.13	HAL_RCC_GetPCLK1Freq	373
33.3.14	HAL_RCC_GetPCLK2Freq	373
33.3.15	HAL_RCC_GetOscConfig	373
33.3.16	HAL_RCC_GetClockConfig	373
33.3.17	HAL_RCC_NMI_IRQHandler	374
33.3.18	HAL_RCC_CSSCallback	374
33.4	RCC Firmware driver defines	374
33.4.1	RCC	374
34	HAL RCC Extension Driver	400
34.1	HAL RCC Extension Driver	400
34.2	RCCEX Firmware driver registers structures	400
34.2.1	RCC_PeriphCLKInitTypeDef	400
34.3	RCCEX Firmware driver API description	400
34.3.1	Extended Peripheral Control functions	400
34.3.2	HAL_RCCEX_PeriphCLKConfig	400
34.3.3	HAL_RCCEX_GetPeriphCLKConfig	401
34.3.4	HAL_RCCEX_GetPeriphCLKFreq	401
34.3.5	HAL_RCCEX_EnableLSECSS	401
34.3.6	HAL_RCCEX_DisableLSECSS	402
34.4	RCCEX Firmware driver defines	402
34.4.1	RCCEX	402
35	HAL RTC Generic Driver	408
35.1	HAL RTC Generic Driver	408

35.2	RTC Firmware driver registers structures	408
35.2.1	RTC_InitTypeDef	408
35.2.2	RTC_DateTypeDef	408
35.2.3	RTC_HandleTypeDef	409
35.3	RTC Firmware driver API description	409
35.3.1	Backup Domain Operating Condition	409
35.3.2	Backup Domain Reset	410
35.3.3	Backup Domain Access	410
35.3.4	How to use this driver	410
35.3.5	RTC and low power modes	411
35.3.6	Initialization and de-initialization functions	411
35.3.7	RTC Time and Date functions	411
35.3.8	RTC Alarm functions	412
35.3.9	Peripheral State functions	412
35.3.10	Peripheral Control functions	412
35.3.11	HAL_RTC_Init	412
35.3.12	HAL_RTC_DeInit	412
35.3.13	HAL_RTC_MspInit	413
35.3.14	HAL_RTC_MspDeInit	413
35.3.15	HAL_RTC_SetTime	413
35.3.16	HAL_RTC_GetTime	413
35.3.17	HAL_RTC_SetDate	414
35.3.18	HAL_RTC_GetDate	414
35.3.19	HAL_RTC_SetTime	414
35.3.20	HAL_RTC_SetDate	415
35.3.21	HAL_RTC_GetDate	415
35.3.22	HAL_RTC_GetTime	415
35.3.23	HAL_RTC_SetAlarm	416
35.3.24	HAL_RTC_SetAlarm_IT	416
35.3.25	HAL_RTC_DeactivateAlarm	417
35.3.26	HAL_RTC_GetAlarm	417
35.3.27	HAL_RTC_AlarmIRQHandler	417
35.3.28	HAL_RTC_PollForAlarmAEvent	417
35.3.29	HAL_RTC_AlarmAEventCallback	418
35.3.30	HAL_RTC_DeactivateAlarm	418
35.3.31	HAL_RTC_AlarmIRQHandler	418
35.3.32	HAL_RTC_AlarmAEventCallback	418
35.3.33	HAL_RTC_PollForAlarmAEvent	418
35.3.34	HAL_RTC_SetAlarm	419

35.3.35	HAL_RTC_SetAlarm_IT	419
35.3.36	HAL_RTC_GetAlarm	419
35.3.37	HAL_RTC_WaitForSynchro	420
35.3.38	HAL_RTC_GetState	420
35.3.39	HAL_RTC_WaitForSynchro	420
35.4	RTC Firmware driver defines	421
35.4.1	RTC	421
36	HAL RTC Extension Driver	431
36.1	HAL RTC Extension Driver	431
36.2	RTCEX Firmware driver registers structures	431
36.2.1	RTC_TamperTypeDef	431
36.2.2	RTC_TimeTypeDef	431
36.2.3	RTC_AlarmTypeDef	432
36.3	RTCEX Firmware driver API description	433
36.3.1	How to use this driver	433
36.3.2	RTC TimeStamp and Tamper functions	434
36.3.3	RTC Wake-up functions	434
36.3.4	Extension Peripheral Control functions	434
36.3.5	Extended features functions	435
36.3.6	HAL_RTCEX_SetTimeStamp	435
36.3.7	HAL_RTCEX_SetTimeStamp_IT	435
36.3.8	HAL_RTCEX_DeactivateTimeStamp	436
36.3.9	HAL_RTCEX_GetTimeStamp	436
36.3.10	HAL_RTCEX_SetTamper	436
36.3.11	HAL_RTCEX_SetTamper_IT	437
36.3.12	HAL_RTCEX_DeactivateTamper	437
36.3.13	HAL_RTCEX_TamperTimeStampIRQHandler	437
36.3.14	HAL_RTCEX_TimeStampEventCallback	437
36.3.15	HAL_RTCEX_Tamper1EventCallback	437
36.3.16	HAL_RTCEX_Tamper2EventCallback	438
36.3.17	HAL_RTCEX_Tamper3EventCallback	438
36.3.18	HAL_RTCEX_PollForTimeStampEvent	438
36.3.19	HAL_RTCEX_PollForTamper1Event	438
36.3.20	HAL_RTCEX_PollForTamper2Event	438
36.3.21	HAL_RTCEX_PollForTamper3Event	439
36.3.22	HAL_RTCEX_SetWakeUpTimer	439
36.3.23	HAL_RTCEX_SetWakeUpTimer_IT	439
36.3.24	HAL_RTCEX_DeactivateWakeUpTimer	439

36.3.25	HAL_RTCEx_GetWakeUpTimer	440
36.3.26	HAL_RTCEx_WakeUpTimerIRQHandler	440
36.3.27	HAL_RTCEx_WakeUpTimerEventCallback	440
36.3.28	HAL_RTCEx_PollForWakeUpTimerEvent	440
36.3.29	HAL_RTCEx_BKUPWrite	440
36.3.30	HAL_RTCEx_BKUPRead	441
36.3.31	HAL_RTCEx_SetCoarseCalib	441
36.3.32	HAL_RTCEx_DeactivateCoarseCalib	441
36.3.33	HAL_RTCEx_SetSmoothCalib	441
36.3.34	HAL_RTCEx_SetSynchroShift	442
36.3.35	HAL_RTCEx_SetCalibrationOutPut	442
36.3.36	HAL_RTCEx_DeactivateCalibrationOutPut	443
36.3.37	HAL_RTCEx_SetRefClock	443
36.3.38	HAL_RTCEx_DeactivateRefClock	443
36.3.39	HAL_RTCEx_EnableBypassShadow	443
36.3.40	HAL_RTCEx_DisableBypassShadow	444
36.3.41	HAL_RTCEx_AlarmBEventCallback	444
36.3.42	HAL_RTCEx_PollForAlarmBEvent	444
36.4	RTCEx Firmware driver defines	444
36.4.1	RTCEx	444
37	HAL SD Generic Driver	468
37.1	HAL SD Generic Driver	468
37.2	SD Firmware driver registers structures	468
37.2.1	SD_HandleTypeDef	468
37.2.2	HAL_SD_CSDef	469
37.2.3	HAL_SD_CIDTypeDef	471
37.2.4	HAL_SD_CardStatusTypeDef	472
37.2.5	HAL_SD_CardInfoTypeDef	472
37.3	SD Firmware driver API description	473
37.3.1	How to use this driver	473
37.3.2	Initialization and de-initialization functions	475
37.3.3	IO operation functions	475
37.3.4	Peripheral Control functions	476
37.3.5	Peripheral State functions	476
37.3.6	HAL_SD_Init	476
37.3.7	HAL_SD_DeInit	476
37.3.8	HAL_SD_MspInit	476
37.3.9	HAL_SD_MspDeInit	476

37.3.10	HAL_SD_ReadBlocks	477
37.3.11	HAL_SD_WriteBlocks.....	477
37.3.12	HAL_SD_ReadBlocks_DMA	477
37.3.13	HAL_SD_WriteBlocks_DMA	478
37.3.14	HAL_SD_CheckReadOperation.....	478
37.3.15	HAL_SD_CheckWriteOperation	478
37.3.16	HAL_SD_Erase	478
37.3.17	HAL_SD_IRQHandler.....	479
37.3.18	HAL_SD_XferCpltCallback.....	479
37.3.19	HAL_SD_XferErrorCallback	479
37.3.20	HAL_SD_DMA_RxCpltCallback.....	479
37.3.21	HAL_SD_DMA_RxErrorCallback	479
37.3.22	HAL_SD_DMA_TxCpltCallback	480
37.3.23	HAL_SD_DMA_TxErrorCallback.....	480
37.3.24	HAL_SD_Get_CardInfo	480
37.3.25	HAL_SD_WideBusOperation_Config	480
37.3.26	HAL_SD_StopTransfer.....	480
37.3.27	HAL_SD_HighSpeed.....	481
37.3.28	HAL_SD_SendSDStatus	481
37.3.29	HAL_SD_GetStatus.....	481
37.3.30	HAL_SD_GetCardStatus.....	481
37.4	SD Firmware driver defines.....	482
37.4.1	SD	482
38	HAL SMARTCARD Generic Driver.....	495
38.1	HAL SMARTCARD Generic Driver	495
38.2	SMARTCARD Firmware driver registers structures	495
38.2.1	SMARTCARD_InitTypeDef	495
38.2.2	SMARTCARD_HandleTypeDef.....	496
38.3	SMARTCARD Firmware driver API description.....	497
38.3.1	How to use this driver	497
38.3.2	Initialization and Configuration functions.....	499
38.3.3	IO operation functions	500
38.3.4	Peripheral State and Errors functions	501
38.3.5	HAL_SMARTCARD_Init.....	501
38.3.6	HAL_SMARTCARD_DeInit	501
38.3.7	HAL_SMARTCARD_MspInit	501
38.3.8	HAL_SMARTCARD_MspDeInit	501
38.3.9	HAL_SMARTCARD_Transmit.....	502

38.3.10	HAL_SMARTCARD_Receive.....	502
38.3.11	HAL_SMARTCARD_Transmit_IT	502
38.3.12	HAL_SMARTCARD_Receive_IT	502
38.3.13	HAL_SMARTCARD_Transmit_DMA.....	503
38.3.14	HAL_SMARTCARD_Receive_DMA.....	503
38.3.15	HAL_SMARTCARD_IRQHandler.....	503
38.3.16	HAL_SMARTCARD_TxCpltCallback	504
38.3.17	HAL_SMARTCARD_RxCpltCallback	504
38.3.18	HAL_SMARTCARD_ErrorCallback.....	504
38.3.19	HAL_SMARTCARD_GetState	504
38.3.20	HAL_SMARTCARD_GetError.....	504
38.4	SMARTCARD Firmware driver defines	505
38.4.1	SMARTCARD.....	505
39	HAL SPI Generic Driver.....	516
39.1	HAL SPI Generic Driver	516
39.2	SPI Firmware driver registers structures	516
39.2.1	SPI_InitTypeDef	516
39.2.2	__SPI_HandleTypeDef.....	517
39.3	SPI Firmware driver API description	518
39.3.1	How to use this driver	518
39.3.2	Initialization and de-initialization functions	518
39.3.3	IO operation functions	519
39.3.4	Peripheral State and Errors functions	520
39.3.5	HAL_SPI_Init.....	520
39.3.6	HAL_SPI_DeInit	520
39.3.7	HAL_SPI_MspInit	520
39.3.8	HAL_SPI_MspDeInit.....	520
39.3.9	HAL_SPI_Transmit.....	520
39.3.10	HAL_SPI_Receive.....	521
39.3.11	HAL_SPI_TransmitReceive.....	521
39.3.12	HAL_SPI_Transmit_IT.....	521
39.3.13	HAL_SPI_Receive_IT.....	521
39.3.14	HAL_SPI_TransmitReceive_IT	522
39.3.15	HAL_SPI_Transmit_DMA.....	522
39.3.16	HAL_SPI_Receive_DMA.....	522
39.3.17	HAL_SPI_TransmitReceive_DMA.....	523
39.3.18	HAL_SPI_DMAPause.....	523
39.3.19	HAL_SPI_DMAResume	523

39.3.20	HAL_SPI_DMAStop	523
39.3.21	HAL_SPI_IRQHandler.....	523
39.3.22	HAL_SPI_TxCpltCallback	524
39.3.23	HAL_SPI_RxCpltCallback	524
39.3.24	HAL_SPI_TxRxCpltCallback	524
39.3.25	HAL_SPI_TxHalfCpltCallback	524
39.3.26	HAL_SPI_RxHalfCpltCallback.....	524
39.3.27	HAL_SPI_TxRxHalfCpltCallback.....	525
39.3.28	HAL_SPI_ErrorCallback.....	525
39.3.29	HAL_SPI_GetState.....	525
39.3.30	HAL_SPI_GetError	525
39.4	SPI Firmware driver defines.....	525
39.4.1	SPI	525
40	HAL SRAM Generic Driver	532
40.1	HAL SRAM Generic Driver	532
40.2	SRAM Firmware driver registers structures.....	532
40.2.1	SRAM_HandleTypeDef	532
40.3	SRAM Firmware driver API description.....	532
40.3.1	How to use this driver	532
40.3.2	SRAM Initialization and de_initialization functions	533
40.3.3	SRAM Input and Output functions	533
40.3.4	SRAM Control functions	533
40.3.5	SRAM State functions	534
40.3.6	HAL_SRAM_Init	534
40.3.7	HAL_SRAM_DeInit.....	534
40.3.8	HAL_SRAM_MspInit.....	534
40.3.9	HAL_SRAM_MspDeInit.....	534
40.3.10	HAL_SRAM_DMA_XferCpltCallback	535
40.3.11	HAL_SRAM_DMA_XferErrorCallback.....	535
40.3.12	HAL_SRAM_Read_8b.....	535
40.3.13	HAL_SRAM_Write_8b.....	535
40.3.14	HAL_SRAM_Read_16b.....	535
40.3.15	HAL_SRAM_Write_16b.....	536
40.3.16	HAL_SRAM_Read_32b.....	536
40.3.17	HAL_SRAM_Write_32b.....	536
40.3.18	HAL_SRAM_Read_DMA.....	537
40.3.19	HAL_SRAM_Write_DMA.....	537
40.3.20	HAL_SRAM_WriteOperation_Enable.....	537

40.3.21	HAL_SRAM_WriteOperation_Disable	537
40.3.22	HAL_SRAM_GetState	537
40.4	SRAM Firmware driver defines	538
40.4.1	SRAM	538
41	HAL TIM Generic Driver	539
41.1	HAL TIM Generic Driver	539
41.2	TIM Firmware driver registers structures	539
41.2.1	TIM_Base_InitTypeDef	539
41.2.2	TIM_OC_InitTypeDef	539
41.2.3	TIM_OnePulse_InitTypeDef	540
41.2.4	TIM_IC_InitTypeDef	540
41.2.5	TIM_Encoder_InitTypeDef	541
41.2.6	TIM_ClockConfigTypeDef	542
41.2.7	TIM_ClearInputConfigTypeDef	542
41.2.8	TIM_SlaveConfigTypeDef	543
41.2.9	TIM_HandleTypeDef	543
41.3	TIM Firmware driver API description	544
41.3.1	TIMER Generic features	544
41.3.2	How to use this driver	544
41.3.3	Time Base functions	545
41.3.4	Time Output Compare functions	545
41.3.5	Time PWM functions	546
41.3.6	Time Input Capture functions	546
41.3.7	Time One Pulse functions	547
41.3.8	Time Encoder functions	547
41.3.9	IRQ handler management	548
41.3.10	Peripheral Control functions	548
41.3.11	TIM Callbacks functions	548
41.3.12	Peripheral State functions	549
41.3.13	HAL_TIM_Base_Init	549
41.3.14	HAL_TIM_Base_DeInit	549
41.3.15	HAL_TIM_Base_MspInit	549
41.3.16	HAL_TIM_Base_MspDeInit	549
41.3.17	HAL_TIM_Base_Start	550
41.3.18	HAL_TIM_Base_Stop	550
41.3.19	HAL_TIM_Base_Start_IT	550
41.3.20	HAL_TIM_Base_Stop_IT	550
41.3.21	HAL_TIM_Base_Start_DMA	550

41.3.22	HAL_TIM_Base_Stop_DMA.....	551
41.3.23	HAL_TIM_OC_Init	551
41.3.24	HAL_TIM_OC_DeInit.....	551
41.3.25	HAL_TIM_OC_MspInit	551
41.3.26	HAL_TIM_OC_MspDeInit.....	551
41.3.27	HAL_TIM_OC_Start	551
41.3.28	HAL_TIM_OC_Stop.....	552
41.3.29	HAL_TIM_OC_Start_IT	552
41.3.30	HAL_TIM_OC_Stop_IT	552
41.3.31	HAL_TIM_OC_Start_DMA	553
41.3.32	HAL_TIM_OC_Stop_DMA	553
41.3.33	HAL_TIM_PWM_Init.....	553
41.3.34	HAL_TIM_PWM_DeInit	553
41.3.35	HAL_TIM_PWM_MspInit.....	554
41.3.36	HAL_TIM_PWM_MspDeInit	554
41.3.37	HAL_TIM_PWM_Start.....	554
41.3.38	HAL_TIM_PWM_Stop	554
41.3.39	HAL_TIM_PWM_Start_IT	554
41.3.40	HAL_TIM_PWM_Stop_IT	555
41.3.41	HAL_TIM_PWM_Start_DMA	555
41.3.42	HAL_TIM_PWM_Stop_DMA	555
41.3.43	HAL_TIM_IC_Init	556
41.3.44	HAL_TIM_IC_DeInit	556
41.3.45	HAL_TIM_IC_MspInit	556
41.3.46	HAL_TIM_IC_MspDeInit.....	556
41.3.47	HAL_TIM_IC_Start	556
41.3.48	HAL_TIM_IC_Stop	556
41.3.49	HAL_TIM_IC_Start_IT	557
41.3.50	HAL_TIM_IC_Stop_IT	557
41.3.51	HAL_TIM_IC_Start_DMA	557
41.3.52	HAL_TIM_IC_Stop_DMA	558
41.3.53	HAL_TIM_OnePulse_Init.....	558
41.3.54	HAL_TIM_OnePulse_DeInit	558
41.3.55	HAL_TIM_OnePulse_MspInit.....	558
41.3.56	HAL_TIM_OnePulse_MspDeInit	559
41.3.57	HAL_TIM_OnePulse_Start.....	559
41.3.58	HAL_TIM_OnePulse_Stop	559
41.3.59	HAL_TIM_OnePulse_Start_IT.....	559

41.3.60	HAL_TIM_OnePulse_Stop_IT	559
41.3.61	HAL_TIM_Encoder_Init	560
41.3.62	HAL_TIM_Encoder_DeInit	560
41.3.63	HAL_TIM_Encoder_MspInit	560
41.3.64	HAL_TIM_Encoder_MspDeInit.....	560
41.3.65	HAL_TIM_Encoder_Start	560
41.3.66	HAL_TIM_Encoder_Stop	561
41.3.67	HAL_TIM_Encoder_Start_IT	561
41.3.68	HAL_TIM_Encoder_Stop_IT	561
41.3.69	HAL_TIM_Encoder_Start_DMA	562
41.3.70	HAL_TIM_Encoder_Stop_DMA	562
41.3.71	HAL_TIM_IRQHandler	562
41.3.72	HAL_TIM_OC_ConfigChannel.....	562
41.3.73	HAL_TIM_IC_ConfigChannel.....	563
41.3.74	HAL_TIM_PWM_ConfigChannel.....	563
41.3.75	HAL_TIM_OnePulse_ConfigChannel.....	563
41.3.76	HAL_TIM_DMABurst_WriteStart.....	564
41.3.77	HAL_TIM_DMABurst_WriteStop	564
41.3.78	HAL_TIM_DMABurst_ReadStart.....	565
41.3.79	HAL_TIM_DMABurst_ReadStop.....	566
41.3.80	HAL_TIM_GenerateEvent	566
41.3.81	HAL_TIM_ConfigOCrefClear.....	566
41.3.82	HAL_TIM_ConfigClockSource	566
41.3.83	HAL_TIM_ConfigTI1Input.....	567
41.3.84	HAL_TIM_SlaveConfigSynchronization	567
41.3.85	HAL_TIM_SlaveConfigSynchronization_IT	567
41.3.86	HAL_TIM_ReadCapturedValue.....	568
41.3.87	HAL_TIM_PeriodElapsedCallback	568
41.3.88	HAL_TIM_OC_DelayElapsedCallback.....	568
41.3.89	HAL_TIM_IC_CaptureCallback	568
41.3.90	HAL_TIM_PWM_PulseFinishedCallback	568
41.3.91	HAL_TIM_TriggerCallback	569
41.3.92	HAL_TIM_ErrorCallback.....	569
41.3.93	HAL_TIM_Base_GetState	569
41.3.94	HAL_TIM_OC_GetState.....	569
41.3.95	HAL_TIM_PWM_GetState	569
41.3.96	HAL_TIM_IC_GetState.....	569
41.3.97	HAL_TIM_OnePulse_GetState	570
41.3.98	HAL_TIM_Encoder_GetState.....	570

41.3.99	TIM_DMAError	570
41.3.100	TIM_DMADelayPulseCplt.....	570
41.3.101	TIM_DMACaptureCplt	570
41.4	TIM Firmware driver defines.....	571
41.4.1	TIM.....	571
42	HAL TIM Extension Driver.....	589
42.1	HAL TIM Extension Driver.....	589
42.2	TIMEx Firmware driver registers structures.....	589
42.2.1	TIM_MasterConfigTypeDef	589
42.3	TIMEx Firmware driver API description.....	589
42.3.1	TIMER Extended features	589
42.3.2	Peripheral Control functions	589
42.3.3	HAL_TIMEx_MasterConfigSynchronization	589
42.3.4	HAL_TIMEx_RemapConfig	590
42.4	TIMEx Firmware driver defines	591
42.4.1	TIMEx	591
43	HAL UART Generic Driver.....	593
43.1	HAL UART Generic Driver	593
43.2	UART Firmware driver registers structures	593
43.2.1	UART_InitTypeDef	593
43.2.2	UART_HandleTypeDef.....	594
43.3	UART Firmware driver API description	595
43.3.1	How to use this driver	595
43.3.2	Initialization and Configuration functions.....	597
43.3.3	IO operation functions	597
43.3.4	Peripheral Control functions	598
43.3.5	Peripheral State and Errors functions	599
43.3.6	HAL_UART_Init	599
43.3.7	HAL_HalfDuplex_Init	599
43.3.8	HAL_LIN_Init	600
43.3.9	HAL_MultiProcessor_Init	600
43.3.10	HAL_UART_DeInit	600
43.3.11	HAL_UART_MspInit	600
43.3.12	HAL_UART_MspDeInit.....	601
43.3.13	HAL_UART_Transmit.....	601
43.3.14	HAL_UART_Receive.....	601
43.3.15	HAL_UART_Transmit_IT.....	601

43.3.16	HAL_UART_Receive_IT	602
43.3.17	HAL_UART_Transmit_DMA	602
43.3.18	HAL_UART_Receive_DMA	602
43.3.19	HAL_UART_DMAPause	602
43.3.20	HAL_UART_DMAResume	603
43.3.21	HAL_UART_DMASStop	603
43.3.22	HAL_UART_IRQHandler	603
43.3.23	HAL_UART_TxCpltCallback	603
43.3.24	HAL_UART_TxHalfCpltCallback	604
43.3.25	HAL_UART_RxCpltCallback	604
43.3.26	HAL_UART_RxHalfCpltCallback	604
43.3.27	HAL_UART_ErrorCallback	604
43.3.28	HAL_LIN_SendBreak	604
43.3.29	HAL_MultiProcessor_EnterMuteMode	605
43.3.30	HAL_MultiProcessor_ExitMuteMode	605
43.3.31	HAL_HalfDuplex_EnableTransmitter	605
43.3.32	HAL_HalfDuplex_EnableReceiver	605
43.3.33	HAL_UART_GetState	605
43.3.34	HAL_UART_GetError	606
43.4	UART Firmware driver defines	606
43.4.1	UART	606
44	HAL USART Generic Driver	618
44.1	HAL USART Generic Driver	618
44.2	USART Firmware driver registers structures	618
44.2.1	USART_InitTypeDef	618
44.2.2	USART_HandleTypeDef	619
44.3	USART Firmware driver API description	620
44.3.1	How to use this driver	620
44.3.2	Initialization and Configuration functions	622
44.3.3	IO operation functions	622
44.3.4	Peripheral State and Errors functions	623
44.3.5	HAL_USART_Init	624
44.3.6	HAL_USART_DeInit	624
44.3.7	HAL_USART_MspInit	624
44.3.8	HAL_USART_MspDeInit	624
44.3.9	HAL_USART_Transmit	624
44.3.10	HAL_USART_Receive	625
44.3.11	HAL_USART_TransmitReceive	625

44.3.12	HAL_USART_Transmit_IT	625
44.3.13	HAL_USART_Receive_IT	626
44.3.14	HAL_USART_TransmitReceive_IT	626
44.3.15	HAL_USART_Transmit_DMA	626
44.3.16	HAL_USART_Receive_DMA	627
44.3.17	HAL_USART_TransmitReceive_DMA	627
44.3.18	HAL_USART_DMAPause	627
44.3.19	HAL_USART_DMAResume	627
44.3.20	HAL_USART_DMAStop	628
44.3.21	HAL_USART_IRQHandler	628
44.3.22	HAL_USART_TxCpltCallback	628
44.3.23	HAL_USART_TxHalfCpltCallback	628
44.3.24	HAL_USART_RxCpltCallback	629
44.3.25	HAL_USART_RxHalfCpltCallback	629
44.3.26	HAL_USART_TxRxCpltCallback	629
44.3.27	HAL_USART_ErrorCallback	629
44.3.28	HAL_USART_GetState	629
44.3.29	HAL_USART_GetError	630
44.4	USART Firmware driver defines	630
44.4.1	USART	630
45	HAL WWDG Generic Driver	639
45.1	HAL WWDG Generic Driver	639
45.2	WWDG Firmware driver registers structures	639
45.2.1	WWDG_InitTypeDef	639
45.2.2	WWDG_HandleTypeDef	639
45.3	WWDG Firmware driver API description	640
45.3.1	WWDG specific features	640
45.3.2	How to use this driver	640
45.3.3	Initialization and de-initialization functions	640
45.3.4	IO operation functions	641
45.3.5	Peripheral State functions	641
45.3.6	HAL_WWDG_Init	641
45.3.7	HAL_WWDG_DeInit	641
45.3.8	HAL_WWDG_MspInit	642
45.3.9	HAL_WWDG_MspDeInit	642
45.3.10	HAL_WWDG_WakeupCallback	642
45.3.11	HAL_WWDG_Start	642
45.3.12	HAL_WWDG_Start_IT	642

45.3.13	HAL_WWDG_Refresh.....	643
45.3.14	HAL_WWDG_IRQHandler	643
45.3.15	HAL_WWDG_WakeupCallback	643
45.3.16	HAL_WWDG_GetState	643
45.4	WWDG Firmware driver defines.....	644
45.4.1	WWDG.....	644
46	FAQs.....	648
47	Revision history	652

List of tables

Table 1: Acronyms and definitions.....	37
Table 2: HAL drivers files.....	39
Table 3: User-application files	40
Table 4: APIs classification	45
Table 5: List of devices supported by HAL drivers	46
Table 6: HAL API naming rules	50
Table 7: Macros handling interrupts and specific clock configurations	51
Table 8: Callback functions.....	52
Table 9: HAL generic APIs	53
Table 10: HAL extension APIs	54
Table 11: Define statements used for HAL configuration	59
Table 12: Description of GPIO_InitTypeDef structure	61
Table 13: Description of EXTI configuration macros	63
Table 14: MSP functions.....	68
Table 15: Timeout values	71
Table 16: Redirection of COMP outputs to embedded timers	122
Table 17: COMP Inputs for the STM32L1xx devices	122
Table 18: IRDA frame formats	271
Table 19: OPAMPs inverting/non-inverting inputs for STM32L1 devices.....	317
Table 20: OPAMP outputs for STM32L1 devices	317
Table 21: Number of wait states (WS) according to CPU clock (HCLK) frequency	369
Table 22: Clock frequency versus product voltage range	369
Table 23: Smartcard frame formats	499
Table 24: UART frame formats.....	597
Table 25: USART frame formats	622
Table 26: Document revision history	652

List of figures

Figure 1: Example of project template	42
Figure 2: Adding device-specific functions	55
Figure 3: Adding family-specific functions	55
Figure 4: Adding new peripherals	56
Figure 5: Updating existing APIs	56
Figure 6: File inclusion model	57
Figure 7: HAL driver model	66

1 Acronyms and definitions

Table 1: Acronyms and definitions

Acronym	Definition
ADC	Analog-to-digital converter
ANSI	American National Standards Institute
API	Application Programming Interface
BSP	Board Support Package
COMP	Comparator
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRYP	Cryptographic processor unit
CRC	CRC calculation unit
DAC	Digital to analog converter
DMA	Direct Memory Access
EXTI	External interrupt/event controller
FLASH	Flash memory
GPIO	General purpose I/Os
HAL	Hardware abstraction layer
I2C	Inter-integrated circuit
I2S	Inter-integrated sound
IRDA	InfraRed Data Association
IWDG	Independent watchdog
LCD	Liquid Crystal Display Controller
MSP	MCU Specific Package
NVIC	Nested Vectored Interrupt Controller
PCD	USB Peripheral Controller Driver
PWR	Power controller
RCC	Reset and clock controller
RNG	Random Number Generator
RTC	Real-time clock
SD	Secure Digital
SRAM	SRAM external memory
SMARTCARD	Smartcard IC
SPI	Serial Peripheral interface
SysTick	System tick timer
TIM	Advanced-control, general-purpose or basic timer
TSC	Touch Sensing Controller

Acronym	Definition
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous receiver/transmitter
WWDG	Window watchdog
USB	Universal Serial Bus
PPP	STM32 peripheral or block

2 Overview of HAL drivers

The HAL drivers were designed to offer a rich set of APIs and to interact easily with the application upper layers.

Each driver consists of a set of functions covering the most common peripheral features. The development of each driver is driven by a common API which standardizes the driver structure, the functions and the parameter names.

The HAL drivers consist of a set of driver modules, each module being linked to a standalone peripheral. However, in some cases, the module is linked to a peripheral functional mode. As an example, several modules exist for the USART peripheral: UART driver module, USART driver module, SMARTCARD driver module and IRDA driver module.

The HAL main features are the following:

- Cross-family portable set of APIs covering the common peripheral features as well as extension APIs in case of specific peripheral features.
- Three API programming models: polling, interrupt and DMA.
- APIs are RTOS compliant:
 - Fully reentrant APIs
 - Systematic usage of timeouts in polling mode.
- Peripheral multi-instance support allowing concurrent API calls for multiple instances of a given peripheral (USART1, USART2...)
- All HAL APIs implement user-callback functions mechanism:
 - Peripheral Init/Delinit HAL APIs can call user-callback functions to perform peripheral system level Initialization/De-Initialization (clock, GPIOs, interrupt, DMA)
 - Peripherals interrupt events
 - Error events.
- Object locking mechanism: safe hardware access to prevent multiple spurious accesses to shared resources.
- Timeout used for all blocking processes: the timeout can be a simple counter or a timebase.

2.1 HAL and user-application files

2.1.1 HAL driver files

A HAL drivers are composed of the following set of files:

Table 2: HAL drivers files

File	Description
<i>stm32l1xx_hal_ppp.c</i>	Main peripheral/module driver file. It includes the APIs that are common to all STM32 devices. <i>Example: stm32l1xx_hal_adc.c, stm32l1xx_hal_irda.c, ...</i>
<i>stm32l1xx_hal_ppp.h</i>	Header file of the main driver C file It includes common data, handle and enumeration structures, define statements and macros, as well as the exported generic APIs. <i>Example: stm32l1xx_hal_adc.h, stm32l1xx_hal_irda.h, ...</i>

File	Description
<i>stm32l1xx_hal_ppp_ex.c</i>	Extension file of a peripheral/module driver. It includes the specific APIs for a given part number or family, as well as the newly defined APIs that overwrite the default generic APIs if the internal process is implemented in different way. <i>Example: stm32l1xx_hal_adc_ex.c, stm32l1xx_hal_dma_ex.c, ...</i>
<i>stm32l1xx_hal_ppp_ex.h</i>	Header file of the extension C file. It includes the specific data and enumeration structures, define statements and macros, as well as the exported device part number specific APIs <i>Example: stm32l1xx_hal_adc_ex.h, stm32l1xx_hal_dma_ex.h, ...</i>
<i>stm32l1xx_hal.c</i>	This file is used for HAL initialization and contains DBGMCU, Remap and Time Delay based on systick APIs.
<i>stm32l1xx_hal.h</i>	stm32l1xx_hal.c header file
<i>stm32l1xx_hal_msp_template.c</i>	Template file to be copied to the user application folder. It contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<i>stm32l1xx_hal_conf_template.h</i>	Template file allowing to customize the drivers for a given application.
<i>stm32l1xx_hal_def.h</i>	Common HAL resources such as common define statements, enumerations, structures and macros.

2.1.2 User-application files

The minimum files required to build an application using the HAL are listed in the table below:

Table 3: User-application files

File	Description
<i>system_stm32l1xx.c</i>	This file contains SystemInit() which is called at startup just after reset and before branching to the main program. It does not configure the system clock at startup (contrary to the standard library). This is to be done using the HAL APIs in the user files. It allows to : <ul style="list-style-type: none"> relocate the vector table in internal SRAM.
<i>startup_stm32l1xx.s</i>	Toolchain specific file that contains reset handler and exception vectors. For some toolchains, it allows adapting the stack/heap size to fit the application requirements.
<i>stm32l1xx_flash.icf</i> (optional)	Linker file for EWARM toolchain allowing mainly to adapt the stack/heap size to fit the application requirements.
<i>stm32l1xx_hal_msp.c</i>	This file contains the MSP initialization and de-initialization (main routine and callbacks) of the peripheral used in the user application.
<i>stm32l1xx_hal_conf.h</i>	This file allows the user to customize the HAL drivers for a specific application. It is not mandatory to modify this configuration. The application can use the default configuration without any modification.

File	Description
<i>stm32l1xx_it.c/.h</i>	<p>This file contains the exceptions handler and peripherals interrupt service routine, and calls HAL_IncTick() at regular time intervals to increment a local variable (declared in <i>stm32l1xx_hal.c</i>) used as HAL timebase. By default, this function is called each 1ms in SysTick ISR. .</p> <p>The PPP_IRQHandler() routine must call HAL_PPP_IRQHandler() if an interrupt based process is used within the application.</p>
<i>main.c/.h</i>	<p>This file contains the main program routine, mainly:</p> <ul style="list-style-type: none"> • the call to HAL_Init() • assert_failed() implementation • system clock configuration • peripheral HAL initialization and user application code.

The STM32Cube package comes with ready-to-use project templates, one for each supported board. Each project contains the files listed above and a preconfigured project for the supported toolchains.

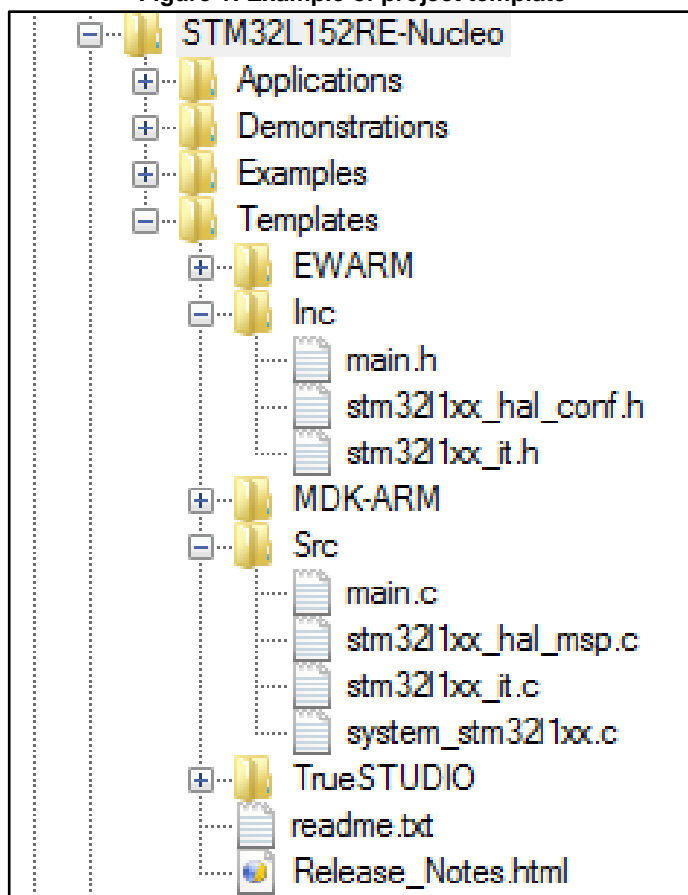
Each project template provides empty main loop function and can be used as a starting point to get familiar with project settings for STM32Cube. Their characteristics are the following:

- It contains sources of HAL, CMSIS and BSP drivers which are the minimal components to develop a code on a given board.
- It contains the include paths for all the firmware components.
- It defines the STM32 device supported, and allows to configure the CMSIS and HAL drivers accordingly.
- It provides ready to use user files preconfigured as defined below:
 - HAL is initialized
 - SysTick ISR implemented for HAL_Delay()
 - System clock configured with the maximum frequency of the device



If an existing project is copied to another location, then include paths must be updated.

Figure 1: Example of project template



2.2 HAL data structures

Each HAL driver can contain the following data structures:

- Peripheral handle structures
- Initialization and configuration structures
- Specific process structures.

2.2.1 Peripheral handle structures

The APIs have a modular generic multi-instance architecture that allows working with several IP instances simultaneously.

PPP_HandleTypeDef *handle is the main structure that is implemented in the HAL drivers. It handles the peripheral/module configuration and registers and embeds all the structures and variables needed to follow the peripheral device flow.

The peripheral handle is used for the following purposes:

- Multi instance support: each peripheral/module instance has its own handle. As a result instance resources are independent.
- Peripheral process intercommunication: the handle is used to manage shared data resources between the process routines.
Example: global pointers, DMA handles, state machine.
- Storage: this handle is used also to manage global variables within a given HAL driver.

An example of peripheral structure is shown below:

```
typedef struct
{
  USART_TypeDef *Instance; /* USART registers base address */
  USART_InitTypeDef Init; /* Usart communication parameters */
  uint8_t *pTxBuffPtr; /* Pointer to Usart Tx transfer Buffer */
  uint16_t TxXferSize; /* Usart Tx Transfer size */
  IO uint16_t TxXferCount; /* Usart Tx Transfer Counter */
  uint8_t *pRxBuffPtr; /* Pointer to Usart Rx transfer Buffer */
  uint16_t RxXferSize; /* Usart Rx Transfer size */
  IO uint16_t RxXferCount; /* Usart Rx Transfer Counter */
  DMA_HandleTypeDef *hdmatx; /* Usart Tx DMA Handle parameters */
  DMA_HandleTypeDef *hdmarx; /* Usart Rx DMA Handle parameters */
  HAL_LockTypeDef Lock; /* Locking object */
  IO HAL_USART_StateTypeDef State; /* Usart communication state */
  IO HAL_USART_ErrorTypeDef ErrorCode; /* USART Error code */
}USART_HandleTypeDef;
```



1) The multi-instance feature implies that all the APIs used in the application are re-entrant and avoid using global variables because subroutines can fail to be re-entrant if they rely on a global variable to remain unchanged but that variable is modified when the subroutine is recursively invoked. For this reason, the following rules are respected:

- Re-entrant code does not hold any static (or global) non-constant data: re-entrant functions can work with global data. For example, a re-entrant interrupt service routine can grab a piece of hardware status to work with (e.g. serial port read buffer) which is not only global, but volatile. Still, typical use of static variables and global data is not advised, in the sense that only atomic read-modify-write instructions should be used in these variables. It should not be possible for an interrupt or signal to occur during the execution of such an instruction.
- Reentrant code does not modify its own code.



2) When a peripheral can manage several processes simultaneously using the DMA (full duplex case), the DMA interface handle for each process is added in the PPP_HandleTypeDef.



3) For the shared and system peripherals, no handle or instance object is used. The peripherals concerned by this exception are the following:

- GPIO
- SYSTICK
- NVIC
- PWR
- RCC
- FLASH.

2.2.2 Initialization and configuration structure

These structures are defined in the generic driver header file when it is common to all part numbers. When they can change from one part number to another, the structures are defined in the extension header file for each part number.

```
typedef struct
{
  uint32_t BaudRate; /*!< This member configures the UART communication baudrate.*/
  uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received
  in a frame.*/
  uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.*/
  uint32_t Parity; /*!< Specifies the parity mode. */
  uint32_t Mode; /*!< Specifies whether the Receive or Transmit mode is enabled or
  disabled.*/
  uint32_t HwFlowCtl; /*!< Specifies whether the hardware flow control mode is enabled
  or disabled.*/
  uint32_t OverSampling; /*!< Specifies whether the Over sampling 8 is enabled or
  disabled,
  to achieve higher speed (up to fPCLK/8).*/
}UART_InitTypeDef;
```



The config structure is used to initialize the sub-modules or sub-instances. See below example:

```
HAL_ADC_ConfigChannel (ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef*
sConfig)
```

2.2.3 Specific process structures

The specific process structures are used for specific process (common APIs). They are defined in the generic driver header file.

Example:

```
HAL_PPP_Process (PPP_HandleTypeDef* hadc, PPP_ProcessConfig* sConfig)
```

2.3 API classification

The HAL APIs are classified into three categories:

- **Generic APIs:** common generic APIs applying to all STM32 devices. These APIs are consequently present in the generic HAL drivers files of all STM32 microcontrollers.

```
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_DeInit(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop_IT(ADC_HandleTypeDef* hadc);
void HAL_ADC_IRQHandler(ADC_HandleTypeDef* hadc);
```

- **Extension APIs:** This set of API is divided into two sub-categories :
 - **Family specific APIs:** APIs applying to a given family. They are located in the extension HAL driver file (see example below related to the ADC).

```
HAL_StatusTypeDef HAL_ADCEX_Calibration_Start(ADC_HandleTypeDef* hadc, uint32_t
SingleDiff);
uint32_t HAL_ADCEX_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t
SingleDiff);
```

- **Device part number specific APIs:** These APIs are implemented in the extension file and delimited by specific define statements relative to a given part number.


```
#if defined(STM32L100xBA) || defined (STM32L151xBA) || defined (STM32L152xBA) || \
defined(STM32L100xC) || defined (STM32L151xC) || defined (STM32L152xC) || defined
(STM32L162xC) || \
defined(STM32L151xCA) || defined (STM32L151xD) || defined (STM32L152xCA) || defined
(STM32L152xD) || defined (STM32L162xCA) || defined (STM32L162xD) || \
defined(STM32L151xE) || defined (STM32L152xE) || defined (STM32L162xE)
void HAL_RCCEX_EnableLSECSS(void);
void HAL_RCCEX_DisableLSECSS(void);
#endif /* STM32L100xBA || ... STM32L162xE*/
```

The data structure related to the specific APIs is delimited by the device part number define statement. It is located in the corresponding extension header C file.

The following table summarizes the location of the different categories of HAL APIs in the driver files.

Table 4: APIs classification

	Generic file	Extension file
Common APIs	X	X ⁽¹⁾
Family specific APIs		X
Device specific APIs		X

Notes:

⁽¹⁾In some cases, the implementation for a specific device part number may change . In this case the generic API is declared as weak function in the extension file. The API is implemented again to overwrite the default function



Family specific APIs are only related to a given family. This means that if a specific API is implemented in another family, and the arguments of this latter family are different, additional structures and arguments might need to be added.



The IRQ handlers are used for common and family specific processes.

2.4 Devices supported by HAL drivers

Table 5: List of devices supported by HAL drivers

Files	VALUE LINE			ACCESS LINE							LCD LINE w/o AES							LCD Line w/ AES				
	STM32L100xB	STM32L100xBA	STM32L100xC	STM32L151xB	STM32L151xBA	STM32L151xC	STM32L151xCA	STM32L151xD	STM32L151xDX	STM32L151xE	STM32L152xB	STM32L152xBA	STM32L152xC	STM32L152xCA	STM32L152xD	STM32L152xDX	STM32L152xE	STM32L162xC	STM32L162xCA	STM32L162xD	STM32L162xDX	STM32L162xE
stm32l1xx_hal.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_adc.c stm32l1xx_hal_adc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_adc_ex.c stm32l1xx_hal_adc_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_comp.c stm32l1xx_hal_comp.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_cortex.c stm32l1xx_hal_cortex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_crc.c stm32l1xx_hal_crc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_cryp.c stm32l1xx_hal_cryp.h	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_dac.c stm32l1xx_hal_dac.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_dac_ex.c stm32l1xx_hal_dac_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_dma.c stm32l1xx_hal_dma.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_flash.c stm32l1xx_hal_flash.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_flash_ramfunc.c stm32l1xx_hal_flash_ramfunc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_flash_ex.c stm32l1xx_hal_flash_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Files	VALUE LINE			ACCESS LINE							LCD LINE w/o AES							LCD Line w/ AES				
	STM32L100xB	STM32L100xBA	STM32L100xC	STM32L151xB	STM32L151xBA	STM32L151xC	STM32L151xCA	STM32L151xD	STM32L151xDX	STM32L151xE	STM32L152xB	STM32L152xBA	STM32L152xC	STM32L152xCA	STM32L152xD	STM32L152xDX	STM32L152xE	STM32L162xC	STM32L162xCA	STM32L162xD	STM32L162xDX	STM32L162xE
stm32l1xx_hal_gpio.c stm32l1xx_hal_gpio.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_i2c.c stm32l1xx_hal_i2c.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_i2c_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_i2s.c stm32l1xx_hal_i2s.h	No	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_irda.c stm32l1xx_hal_irda.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_iwdg.c stm32l1xx_hal_iwdg.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_lcd.c stm32l1xx_hal_lcd.h	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_msp_template.c	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stm32l1xx_hal_nor.c stm32l1xx_hal_nor.h	No	No	No	No	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No	No	No	Yes	No	No
stm32l1xx_hal_opamp.c stm32l1xx_hal_opamp.h	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_opamp_ex.c stm32l1xx_hal_opamp_ex.h	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_pcd.c stm32l1xx_hal_pcd.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_pcd_ex.c stm32l1xx_hal_pcd_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_pwr.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_pwr_ex.c stm32l1xx_hal_pwr_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_rcc.c stm32l1xx_hal_rcc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Overview of HAL drivers

UM1816

Files	VALUE LINE			ACCESS LINE							LCD LINE w/o AES							LCD Line w/ AES				
	STM32L100xB	STM32L100xBA	STM32L100xC	STM32L151xB	STM32L151xBA	STM32L151xC	STM32L151xCA	STM32L151xD	STM32L151xDX	STM32L151xE	STM32L152xB	STM32L152xBA	STM32L152xC	STM32L152xCA	STM32L152xD	STM32L152xDX	STM32L152xE	STM32L162xC	STM32L162xCA	STM32L162xD	STM32L162xDX	STM32L162xE
stm32l1xx_hal_rcc_ex.c stm32l1xx_hal_rcc_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_rtc.c stm32l1xx_hal_rtc.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_rtc_ex.c stm32l1xx_hal_rtc_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_sd.c stm32l1xx_hal_sd.h	No	No	No	No	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No	No	No	Yes	No	No
stm32l1xx_hal_smartcard.c stm32l1xx_hal_smartcard.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_spi.c stm32l1xx_hal_spi.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_spi_ex.c	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_sram.c stm32l1xx_hal_sram.h	No	No	No	No	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No	No	No	Yes	No	No
stm32l1xx_hal_tim.c stm32l1xx_hal_tim.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_tim_ex.c stm32l1xx_hal_tim_ex.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_uart.c stm32l1xx_hal_uart.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_usart.c stm32l1xx_hal_usart.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_hal_wwdg.c stm32l1xx_hal_wwdg.h	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
stm32l1xx_ll_fsmc.c stm32l1xx_ll_fsmc.h	No	No	No	No	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No	No	No	Yes	No	No

Files	VALUE LINE			ACCESS LINE							LCD LINE w/o AES							LCD Line w/ AES				
	STM32L100xB	STM32L100xBA	STM32L100xC	STM32L151xB	STM32L151xBA	STM32L151xC	STM32L151xCA	STM32L151xD	STM32L151xDX	STM32L151xE	STM32L152xB	STM32L152xBA	STM32L152xC	STM32L152xCA	STM32L152xD	STM32L152xDX	STM32L152xE	STM32L162xC	STM32L162xCA	STM32L162xD	STM32L162xDX	STM32L162xE
stm32l1xx_ll_sdmmc.c stm32l1xx_ll_sdmmc.h	No	No	No	No	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No	No	No	Yes	No	No

2.5 HAL drivers rules

2.5.1 HAL API naming rules

The following naming rules are used in HAL drivers:

Table 6: HAL API naming rules

	Generic	Family specific	Device specific
File names	<i>stm32l1xx_hal_ppp (c/h)</i>	<i>stm32l1xx_hal_ppp_ex (c/h)</i>	<i>stm32l1xx_hal_ppp_ex (c/h)</i>
Module name	<i>HAL_PPP_MODULE</i>		
Function name	<i>HAL_PPP_Function</i> <i>HAL_PPP_FeatureFunction_MODE</i>	<i>HAL_PPPEX_Function</i> <i>HAL_PPPEX_FeatureFunction_MODE</i>	<i>HAL_PPPEX_Function</i> <i>HAL_PPPEX_FeatureFunction_MODE</i>
Handle name	<i>PPP_HandleTypeDef</i>	NA	NA
Init structure name	<i>PPP_InitTypeDef</i>	NA	<i>PPP_InitTypeDef</i>
Enum name	<i>HAL_PPP_StructnameTypeDef</i>	NA	NA

- The **PPP** prefix refers to the peripheral functional mode and not to the peripheral itself. For example, if the USART, PPP can be USART, IRDA, UART or SMARTCARD depending on the peripheral mode.
- The constants used in one file are defined within this file. A constant used in several files is defined in a header file. All constants are written in uppercase, except for peripheral driver function parameters.
- typedef variable names should be suffixed with *_TypeDef*.
- Registers are considered as constants. In most cases, their name is in uppercase and uses the same acronyms as in the STM32L1xx reference manuals.
- Peripheral registers are declared in the *PPP_TypeDef* structure (e.g. *ADC_TypeDef*) in *stm32l1xxx.h* header file. *stm32l1xxx.h* corresponds to *stm32l100xb.h*, *stm32l100xba.h*, *stm32l100xc.h*, *stm32l151xb.h*, *stm32l151xba.h*, *stm32l151xc.h*, *stm32l151xca.h*, *stm32l151xd.h*, *stm32l151xe.h*, *stm32l151xdx.h*, *stm32l152xb.h*, *stm32l152xba.h*, *stm32l152xc.h*, *stm32l152xca.h*, *stm32l152xd.h*, *stm32l152xe.h*, *stm32l152xdx.h*, *stm32l162xc.h*, *stm32l162xca.h*, *stm32l162xd.h* or *stm32l162xe.h*, *stm32l162xdx.h*.
- Peripheral function names are prefixed by *HAL_*, then the corresponding peripheral acronym in uppercase followed by an underscore. The first letter of each word is in uppercase (e.g. *HAL_UART_Transmit()*). Only one underscore is allowed in a function name to separate the peripheral acronym from the rest of the function name.
- The structure containing the PPP peripheral initialization parameters are named *PPP_InitTypeDef* (e.g. *ADC_InitTypeDef*).
- The structure containing the Specific configuration parameters for the PPP peripheral are named *PPP_xxxxConfTypeDef* (e.g. *ADC_ChannelConfTypeDef*).
- Peripheral handle structures are named *PPP_HandleTypeDef* (e.g. *DMA_HandleTypeDef*).
- The functions used to initialize the PPP peripheral according to parameters specified in *PPP_InitTypeDef* are named *HAL_PPP_Init* (e.g. *HAL_TIM_Init()*).

- The functions used to reset the PPP peripheral registers to their default values are named `PPP_DeInit`, e.g. `TIM_DeInit`.
- The **MODE** suffix refers to the process mode, which can be polling, interrupt or DMA. As an example, when the DMA is used in addition to the native resources, the function should be called: `HAL_PPP_Function_DMA()`.
- The **Feature** prefix should refer to the new feature.
Example: `HAL_ADC_Start()` refers to the injection mode

2.5.2 HAL general naming rules

- For the shared and system peripherals, no handle or instance object is used. This rule applies to the following peripherals:
 - GPIO
 - SYSTICK
 - NVIC
 - RCC
 - FLASH.

Example: The `HAL_GPIO_Init()` requires only the GPIO address and its configuration parameters.

```
HAL_StatusTypeDef HAL_GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef *Init)
{
  /*GPIO Initialization body */
}
```

- The macros that handle interrupts and specific clock configurations are defined in each peripheral/module driver. These macros are exported in the peripheral driver header files so that they can be used by the extension file. The list of these macros is defined below: This list is not exhaustive and other macros related to peripheral features can be added, so that they can be used in the user application.

Table 7: Macros handling interrupts and specific clock configurations

Macros	Description
<code>__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Enables a specific peripheral interrupt
<code>__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)</code>	Disables a specific peripheral interrupt
<code>__HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)</code>	Gets a specific peripheral interrupt status
<code>__HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)</code>	Clears a specific peripheral interrupt status
<code>__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)</code>	Gets a specific peripheral flag status
<code>__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)</code>	Clears a specific peripheral flag status
<code>__HAL_PPP_ENABLE(__HANDLE__)</code>	Enables a peripheral
<code>__HAL_PPP_DISABLE(__HANDLE__)</code>	Disables a peripheral
<code>__HAL_PPP_XXXX (__HANDLE__, __PARAM__)</code>	Specific PPP HAL driver macro
<code>__HAL_PPP_GET_IT_SOURCE (__HANDLE__, __ INTERRUPT __)</code>	Checks the source of specified interrupt

- NVIC and SYSTICK are two ARM Cortex core features. The APIs related to these features are located in the stm32l1xx_hal_cortex.c file.
- When a status bit or a flag is read from registers, it is composed of shifted values depending on the number of read values and of their size. In this case, the returned status width is 32 bits. Example : STATUS = XX | (YY << 16) or STATUS = XX | (YY << 8) | (YY << 16) | (YY << 24)".
- The PPP handles are valid before using the HAL_PPP_Init() API. The init function performs a check before modifying the handle fields.

```
HAL_PPP_Init(PPP_HandleTypeDef)
if(hppp == NULL)
{
return HAL_ERROR;
}
```

- The macros defined below are used:
 - Conditional macro: #define ABS(x) (((x) > 0) ? (x) : -(x))
 - Pseudo-code macro (multiple instructions macro):

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD__, __DMA_HANDLE__) \
do{ \
( __HANDLE__ )-> __PPP_DMA_FIELD__ = &( __DMA_HANDLE__ ); \
( __DMA_HANDLE__ ).Parent = ( __HANDLE__ ); \
} while(0)
```

2.5.3 HAL interrupt handler and callback functions

Besides the APIs, HAL peripheral drivers include:

- HAL_PPP_IRQHandler() peripheral interrupt handler that should be called from stm32l1xx_it.c
- User callback functions.

The user callback functions are defined as empty functions with “weak” attribute. They have to be defined in the user code.

There are three types of user callbacks functions:

- Peripheral system level initialization/ de-Initialization callbacks: HAL_PPP_MspInit() and HAL_PPP_MspDeInit
- Process complete callbacks : HAL_PPP_ProcessCpltCallback
- Error callback: HAL_PPP_ErrorCallback.

Table 8: Callback functions

Callback functions	Example
HAL_PPP_MspInit() / _DeInit()	Ex: HAL_USART_MspInit() Called from HAL_PPP_Init() API function to perform peripheral system level initialization (GPIOs, clock, DMA, interrupt)
HAL_PPP_ProcessCpltCallback	Ex: HAL_USART_TxCpltCallback Called by peripheral or DMA interrupt handler when the process completes
HAL_PPP_ErrorCallback	Ex: HAL_USART_ErrorCallback Called by peripheral or DMA interrupt handler when an error occurs

2.6 HAL generic APIs

The generic APIs provide common generic functions applying to all STM32 devices. They are composed of four APIs groups:

- **Initialization and de-initialization functions:** `HAL_PPP_Init()`, `HAL_PPP_DeInit()`
- **IO operation functions:** `HAL_PPP_Read()`, `HAL_PPP_Write()`, `HAL_PPP_Transmit()`, `HAL_PPP_Receive()`
- **Control functions:** `HAL_PPP_Set ()`, `HAL_PPP_Get ()`.
- **State and Errors functions:** `HAL_PPP_GetState ()`, `HAL_PPP_GetError ()`.

For some peripheral/module drivers, these groups are modified depending on the peripheral/module implementation.

Example: in the timer driver, the API grouping is based on timer features (PWM, OC, IC...).

The initialization and de-initialization functions allow initializing a peripheral and configuring the low-level resources, mainly clocks, GPIO, alternate functions (AF) and possibly DMA and interrupts. The `HAL_DeInit()` function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.

The IO operation functions perform a row access to the peripheral payload data in write and read modes.

The control functions are used to change dynamically the peripheral configuration and set another operating mode.

The peripheral state and errors functions allow retrieving in runtime the peripheral and data flow states, and identifying the type of errors that occurred. The example below is based on the ADC peripheral. The list of generic APIs is not exhaustive. It is only given as an example.

Table 9: HAL generic APIs

Function Group	Common API Name	Description
<i>Initialization group</i>	<code>HAL_ADC_Init()</code>	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	<code>HAL_ADC_DeInit()</code>	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
<i>IO operation group</i>	<code>HAL_ADC_Start ()</code>	This function starts ADC conversions when the polling method is used
	<code>HAL_ADC_Stop ()</code>	This function stops ADC conversions when the polling method is used
	<code>HAL_ADC_PollForConversion()</code>	This function allows waiting for the end of conversions when the polling method is used. In this case, a timeout value is specified by the user according to the application.
	<code>HAL_ADC_Start_IT()</code>	This function starts ADC conversions when the interrupt method is used
	<code>HAL_ADC_Stop_IT()</code>	This function stops ADC conversions when the interrupt method is used
	<code>HAL_ADC_IRQHandler()</code>	This function handles ADC interrupt requests

Function Group	Common API Name	Description
	<i>HAL_ADC_ConvCpltCallback()</i>	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	<i>HAL_ADC_ErrorCallback()</i>	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	<i>HAL_ADC_ConfigChannel()</i>	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	<i>HAL_ADC_AnalogWDGConfig</i>	This function configures the analog watchdog for the selected ADC
State and Errors group	<i>HAL_ADC_GetState()</i>	This function allows getting in runtime the peripheral and the data flow states.
	<i>HAL_ADC_GetError()</i>	This function allows getting in runtime the error that occurred during IT routine

2.7 HAL extension APIs

2.7.1 HAL extension model overview

The extension APIs provide specific functions or overwrite modified APIs for a specific family (series) or specific part number within the same family.

The extension model consists of an additional file, *stm32l1xx_hal_ppp_ex.c*, that includes all the specific functions and define statements (*stm32l1xx_hal_ppp_ex.h*) for a given part number.

Below an example based on the ADC peripheral:

Table 10: HAL extension APIs

Function Group	Common API Name
<i>HAL_ADCEx_CalibrationStart()</i>	This function is used to start the automatic ADC calibration
<i>HAL_ADCEx_Calibration_GetValue()</i>	This function is used to get the ADC calibration factor
<i>HAL_ADCEx_Calibration_SetValue()</i>	This function is used to set the calibration factor to overwrite automatic conversion result

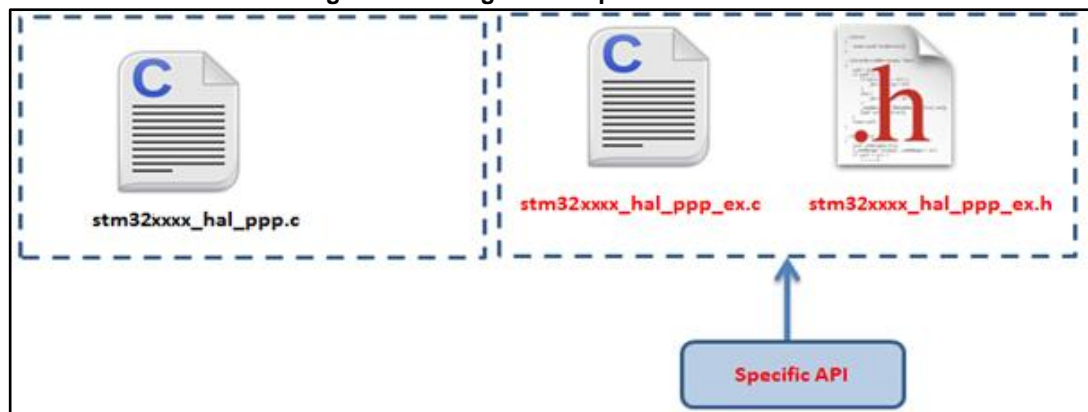
2.7.2 HAL extension model cases

The specific IP features can be handled by the HAL drivers in five different ways. They are described below.

Case1: Adding a part number-specific function

When a new feature specific to a given device is required, the new APIs are added in the *stm32l1xx_hal_ppp_ex.c* extension file. They are named *HAL_PPPEX_Function()*.

Figure 2: Adding device-specific functions



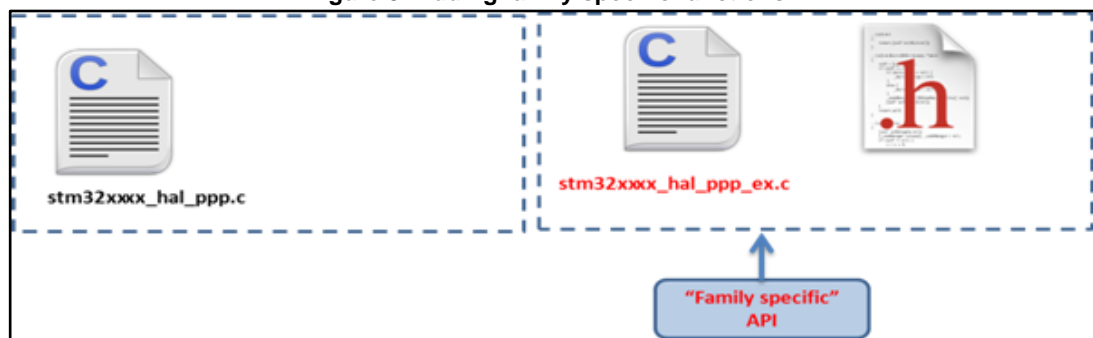
Example: `stm32l1xx_hal_rcc_ex.c/h`

```
#if defined(STM32L100xBA) || defined (STM32L151xBA) || defined (STM32L152xBA) || \
defined(STM32L100xC) || defined (STM32L151xC) || defined (STM32L152xC) || defined
STM32L162xC) || \
defined(STM32L151xCA) || defined (STM32L151xD) || defined (STM32L152xCA) || defined
(STM32L152xD) || defined (STM32L162xCA) || defined (STM32L162xD) ||\
defined(STM32L151xE) || defined (STM32L152xE) || defined (STM32L162xE)
void HAL RCCEX EnableLSECSS(void);
void HAL RCCEX DisableLSECSS(void);
#endif /* STM32L100xBA || ... STM32L162xE*/
```

Case2: Adding a family-specific function

In this case, the API is added in the extension driver C file and named `HAL_PPPEX_Function ()`.

Figure 3: Adding family-specific functions

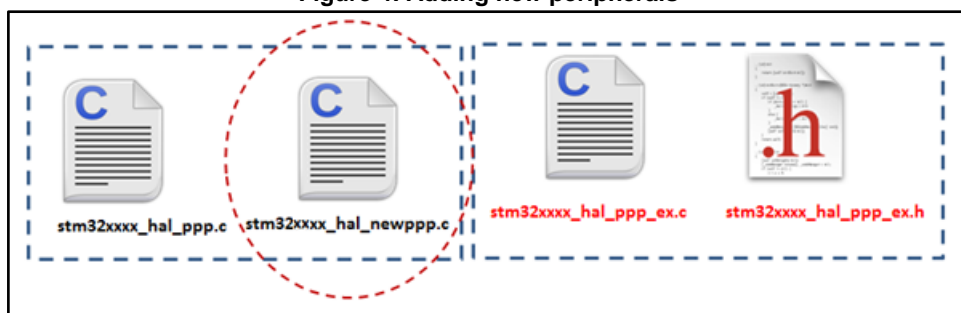


Case3 : Adding a new peripheral (specific to a device belonging to a given family)

When a peripheral which is available only in a specific device is required, the APIs corresponding to this new peripheral/module are added in `stm32l1xx_hal_newppp.c`. However the inclusion of this file is selected in the `stm32lxx_hal_conf.h` using the macro:

```
#define HAL_NEWPPP_MODULE_ENABLED
```

Figure 4: Adding new peripherals

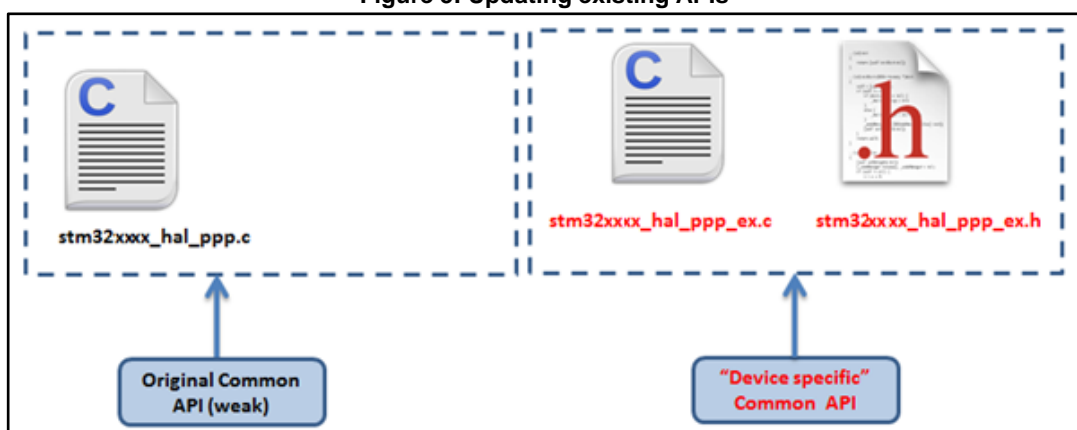


Example: stm32l1xx_hal_lcd.c/h

Case4: Updating existing common APIs

In this case, the routines are defined with the same names in the stm32l1xx_hal_ppp_ex.c extension file, while the generic API is defined as *weak*, so that the compiler will overwrite the original routine by the new defined function.

Figure 5: Updating existing APIs



Case5 : Updating existing data structures

The data structure for a specific device part number (e.g. PPP_InitTypeDef) can have different fields. In this case, the data structure is defined in the extension header file and delimited by the specific part number define statement.

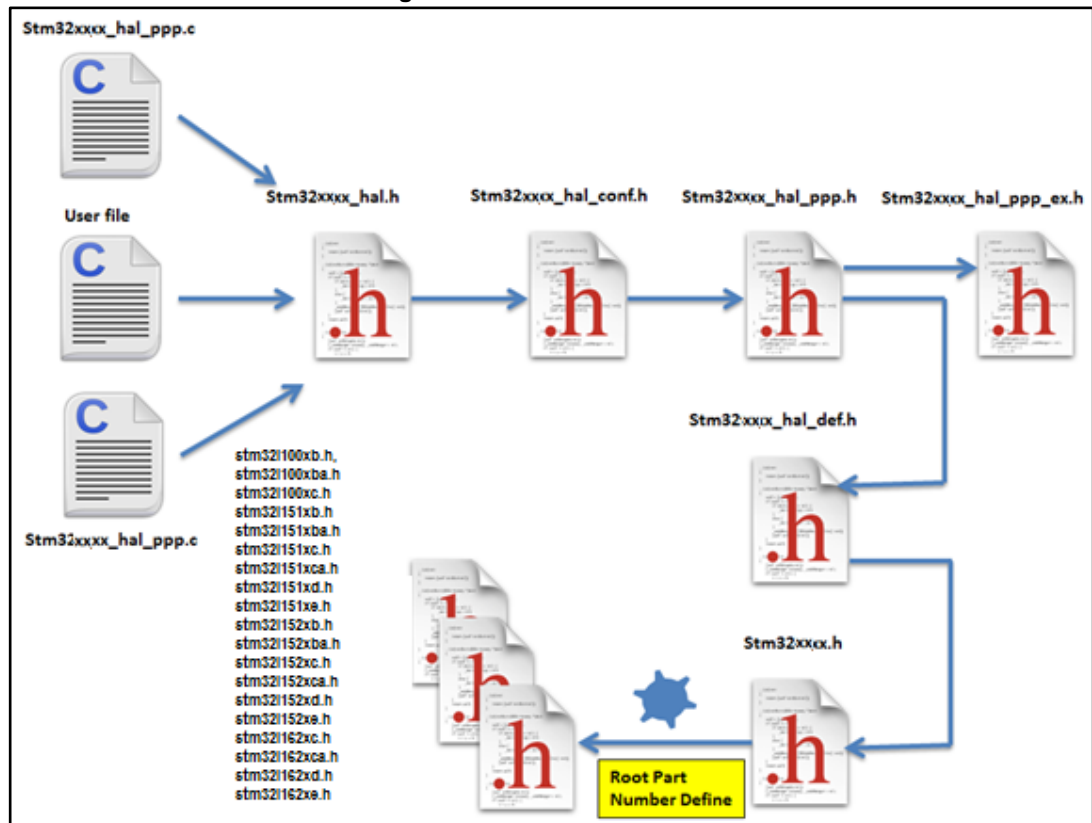
Example:

```
#if defined (STM32L100xB)
typedef struct
{
    (...)
}PPP_InitTypeDef;
#endif /* STM32L100xB */
```

2.8 File inclusion model

The header of the common HAL driver file (stm32l1xx_hal.h) includes the common configurations for the whole HAL library. It is the only header file that is included in the user sources and the HAL C sources files to be able to use the HAL resources.

Figure 6: File inclusion model



A PPP driver is a standalone module which is used in a project. The user must enable the corresponding USE_HAL_PPP_MODULE define statement in the configuration file.

```

/*****
 * @file stm32l1xx_hal_conf.h
 * @author MCD Application Team
 * @version VX.Y.Z * @date dd-mm-yyyy
 * @brief This file contains the modules to be used
 *****/
(...)
#define USE_HAL_USART_MODULE
#define USE_HAL_IRDA_MODULE
#define USE_HAL_DMA_MODULE
#define USE_HAL_RCC_MODULE
(...)

```

2.9 HAL common resources

The common HAL resources, such as common define enumerations, structures and macros, are defined in *stm3211xx_hal_def.h*. The main common define enumeration is *HAL_StatusTypeDef*.

- **HAL Status** The HAL status is used by almost all HAL APIs, except for boolean functions and IRQ handler. It returns the status of the current API operations. It has four possible values as described below:

```
typedef enum
{
  HAL_OK = 0x00,
  HAL_ERROR = 0x01,
  HAL_BUSY = 0x02,
  HAL_TIMEOUT = 0x03
} HAL_StatusTypeDef;
```

- **HAL Locked** The HAL lock is used by all HAL APIs to prevent accessing by accident shared resources.

```
typedef enum
{
  HAL_UNLOCKED = 0x00, /*!<Resources unlocked */
  HAL_LOCKED = 0x01 /*!< Resources locked */
} HAL_LockTypeDef;

In addition to common resources, the stm3211xx_hal_def.h file calls the stm3211xx.h file in CMSIS library to get the data structures and the address mapping for all peripherals:
```

- Declarations of peripheral registers and bits definition.
- Macros to access peripheral registers hardware (Write register, Read register...etc.).

- **Common macros**
 - Macro defining HAL_MAX_DELAY

```
#define HAL_MAX_DELAY 0xFFFFFFFF
```

- Macro linking a PPP peripheral to a DMA structure pointer: `__HAL_LINKDMA();`

```
#define __HAL_LINKDMA(__HANDLE__, __PPP_DMA_FIELD_, __DMA_HANDLE_) \
do{ \
  (__HANDLE__)->__PPP_DMA_FIELD_ = &(__DMA_HANDLE_); \
  (__DMA_HANDLE_).Parent = (__HANDLE_); \
} while(0)
```

2.10 HAL configuration

The configuration file, *stm32l1xx_hal_conf.h*, allows customizing the drivers for the user application. Modifying this configuration is not mandatory: the application can use the default configuration without any modification.

To configure these parameters, the user should enable, disable or modify some options by uncommenting, commenting or modifying the values of the related define statements as described in the table below:

Table 11: Define statements used for HAL configuration

Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	8 000 000 (Hz)
HSE_STARTUP_TIMEOUT	Timeout for HSE start up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)
MSI_VALUE	Defines the Internal Multiple Speed oscillator (MSI) value expressed in Hz.	2 097 000 (Hz)
LSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	32768 (Hz)
LSE_STARTUP_TIMEOUT	Timeout for LSE start up, expressed in ms	5000
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	TRUE
BUFFER_CACHE_ENABLE	Enables buffer cache	FALSE



The *stm32l1xx_hal_conf_template.h* file is located in the HAL drivers *Inc* folder. It should be copied to the user folder, renamed and modified as described above.



By default, the values defined in the *stm32l1xx_hal_conf_template.h* file are the same as the ones used for the examples and demonstrations. All HAL include files are enabled so that they can be used in the user code without modifications.

2.11 HAL system peripheral handling

This chapter gives an overview of how the system peripherals are handled by the HAL drivers. The full API list is provided within each peripheral driver description section.

2.11.1 Clock

Two main functions can be used to configure the system clock:

- `HAL_RCC_OscConfig` (`RCC_OscInitTypeDef *RCC_OscInitStruct`). This function configures/enables multiple clock sources (HSE, HSI, LSE, LSI, PLL).
- `HAL_RCC_ClockConfig` (`RCC_ClkInitTypeDef *RCC_ClkInitStruct`, `uint32_t FLatency`). This function
 - Selects the system clock source
 - Configures AHB, APB1 and APB2 clock dividers
 - Configures the number of Flash memory wait states
 - Updates the SysTick configuration when HCLK clock changes.

Some peripheral clocks are not derived from the system clock (RTC, USB...). In this case, the clock configuration is performed by an extended API defined in `stm32l1xx_hal_rcc_ex.c`: `HAL_RCCEx_PeriphCLKConfig` (`RCC_PeriphCLKInitTypeDef *PeriphClkInit`).

Additional RCC HAL driver functions are available:

- `HAL_RCC_DeInit()` Clock de-init function that return clock configuration to reset state
- Get clock functions that allow retrieving various clock configurations (system clock, HCLK, PCLK1, PCLK2, ...)
- MCO and CSS configuration functions

A set of macros are defined in `stm32l1xx_hal_rcc.h` and `stm32l1xx_hal_rcc_ex.h`. They allow executing elementary operations on RCC block registers, such as peripherals clock gating/reset control:

- `__PPP_CLK_ENABLE/ __PPP_CLK_DISABLE` to enable/disable the peripheral clock
- `__PPP_FORCE_RESET/ __PPP_RELEASE_RESET` to force/release peripheral reset
- `__PPP_CLK_SLEEP_ENABLE/ __PPP_CLK_SLEEP_DISABLE` to enable/disable the peripheral clock during low power (Sleep) mode.

2.11.2 GPIOs

GPIO HAL APIs are the following:

- `HAL_GPIO_Init()` / `HAL_GPIO_DeInit()`
- `HAL_GPIO_ReadPin()` / `HAL_GPIO_WritePin()`
- `HAL_GPIO_TogglePin()`.

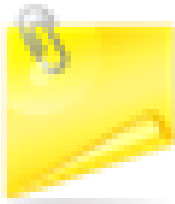
In addition to standard GPIO modes (input, output, analog), pin mode can be configured as EXTI with interrupt or event generation.

When selecting EXTI mode with interrupt generation, the user must call `HAL_GPIO_EXTI_IRQHandler()` from `stm32l1xx_it.c` and implement `HAL_GPIO_EXTI_Callback()`

The table below describes the `GPIO_InitTypeDef` structure field.

Table 12: Description of GPIO_InitTypeDef structure

Structure field	Description
Pin	Specifies the GPIO pins to be configured. Possible values: GPIO_PIN_x or GPIO_PIN_All, where x[0..15]
Mode	Specifies the operating mode for the selected pins: GPIO mode or EXTI mode. Possible values are: <ul style="list-style-type: none"> • <u>GPIO mode</u> <ul style="list-style-type: none"> – GPIO_MODE_INPUT : Input Floating – GPIO_MODE_OUTPUT_PP : Output Push Pull – GPIO_MODE_OUTPUT_OD : Output Open Drain – GPIO_MODE_AF_PP : Alternate Function Push Pull – GPIO_MODE_AF_OD : Alternate Function Open Drain – GPIO_MODE_ANALOG : Analog mode • <u>External Interrupt Mode</u> <ul style="list-style-type: none"> – GPIO_MODE_IT_RISING : Rising edge trigger detection – GPIO_MODE_IT_FALLING : Falling edge trigger detection – GPIO_MODE_IT_RISING_FALLING : Rising/Falling edge trigger detection • <u>External Event Mode</u> <ul style="list-style-type: none"> – GPIO_MODE_EVT_RISING : Rising edge trigger detection – GPIO_MODE_EVT_FALLING : Falling edge trigger detection – GPIO_MODE_EVT_RISING_FALLING: Rising/Falling edge trigger detection
Pull	Specifies the Pull-up or Pull-down activation for the selected pins. Possible values are: GPIO_NOPULL GPIO_PULLUP GPIO_PULLDOWN
Speed	Specifies the speed for the selected pins Possible values are: GPIO_SPEED_VERY_LOW GPIO_SPEED_LOW GPIO_SPEED_MEDIUM GPIO_SPEED_HIGH

Structure field	Description
Alternate	<p>Peripheral to be connected to the selected pins. Possible values: GPIO_AFx_PPP, where AFx: is the alternate function index PPP: is the peripheral instance Example: use GPIO_AF1_TIM2 to connect TIM2 IOs on AF1. These values are defined in the GPIO extended driver, since the AF mapping may change between product lines.</p>  <p>Refer to the “Alternate function mapping” table in the datasheets for the detailed description of the system and peripheral I/O alternate functions.</p>

Please find below typical GPIO configuration examples:

- Configuring GPIOs as output push-pull to drive external LEDs

```
GPIO_InitStruct.Pin = GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_MEDIUM;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
```

- Configuring PA0 as external interrupt with falling edge sensitivity:

```
GPIO_InitStructure.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Pin = GPIO_PIN_0;
HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
```

- Configuring USART1 Tx (PA9, mapped on AF4) as alternate function:

```
GPIO_InitStruct.Pin = GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
GPIO_InitStruct.Alternate = GPIO_AF4_USART1;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

2.11.3 Cortex NVIC and SysTick timer

The Cortex HAL driver, `stm32l1xx_hal_cortex.c`, provides APIs to handle NVIC and SysTick. The supported APIs include:

- HAL_NVIC_SetPriority()
- HAL_NVIC_EnableIRQ()/HAL_NVIC_DisableIRQ()
- HAL_NVIC_SystemReset()
- HAL_SYSTICK_IRQHandler()
- HAL_NVIC_GetPendingIRQ() / HAL_NVIC_SetPendingIRQ () / HAL_NVIC_ClearPendingIRQ()
- HAL_SYSTICK_Config()
- HAL_SYSTICK_CLKSourceConfig()
- HAL_SYSTICK_Callback()

2.11.4 PWR

The PWR HAL driver handles power management. The features shared between all STM32 Series are listed below:

- PVD configuration, enabling/disabling and interrupt handling
 - HAL_PWR_PVDConfig()
 - HAL_PWR_EnablePVD() / HAL_PWR_DisablePVD()
 - HAL_PWR_PVD_IRQHandler()
 - HAL_PWR_PVDCallback()
- Wakeup pin configuration
 - HAL_PWR_EnableWakeUpPin() / HAL_PWR_DisableWakeUpPin()
- Low power mode entry
 - HAL_PWR_EnterSLEEPMode()
 - HAL_PWR_EnterSTOPMode()
 - HAL_PWR_EnterSTANDBYMode()

Depending on the STM32 Series, extension functions are available in stm32l1xx_hal_pwr_ex. Here are a few examples (the list is not exhaustive)

- Ultra low power mode control
 - HAL_PWREx_EnableUltraLowPower() / HAL_PWREx_DisableUltraLowPower()
 - HAL_PWREx_EnableLowPowerRunMode() / HAL_PWREx_DisableLowPowerRunMode()

2.11.5 EXTI

The EXTI is not considered as a standalone peripheral but rather as a service used by other peripheral. As a result there are no EXTI APIs but each peripheral HAL driver implements the associated EXTI configuration and EXTI function are implemented as macros in its header file.

The first 16 EXTI lines connected to the GPIOs are managed within the GPIO driver. The GPIO_InitTypeDef structure allows configuring an I/O as external interrupt or external event.

The EXTI lines connected internally to the PVD, RTC, USB, and COMP are configured within the HAL drivers of these peripheral through the macros given in the table below. The EXTI internal connections depend on the targeted STM32 microcontroller (refer to the product datasheet for more details):

Table 13: Description of EXTI configuration macros

Macros	Description
PPP_EXTI_LINE_FUNCTION	Defines the EXTI line connected to the internal peripheral. Example: #define PWR_EXTI_LINE_PVD ((uint32_t)0x00010000) /*!<External interrupt line 16 Connected to the PVD EXTI Line */
__HAL_PPP_EXTI_ENABLE_IT(__EXTI_LINE__)	Enables a given EXTI line Example: __HAL_PVD_EXTI_ENABLE_IT(PWR_EXTI_LINE_PVD)
__HAL_PPP_EXTI_DISABLE_IT(__EXTI_LINE__)	Disables a given EXTI line. Example: __HAL_PVD_EXTI_DISABLE_IT(PWR_EXTI_LINE_PVD)

Macros	Description
<code>__HAL_PPP_EXTI_GET_FLAG(__EXTI_LINE__)</code>	Gets a given EXTI line interrupt flag pending bit status. Example: <code>__HAL_PVD_EXTI_GET_FLAG(PWR_EXTI_LINE_PVD)</code>
<code>__HAL_PPP_EXTI_CLEAR_FLAG(__EXTI_LINE__)</code>	Clears a given EXTI line interrupt flag pending bit. Example; <code>__HAL_PVD_EXTI_CLEAR_FLAG(PWR_EXTI_LINE_PVD)</code>
<code>__HAL_PPP_EXTI_GENERATE_SWIT(__EXTI_LINE__)</code>	Generates a software interrupt for a given EXTI line. Example: <code>__HAL_PVD_EXTI_GENERATE_SWIT(PWR_EXTI_LINE_PVD)</code>

If the EXTI interrupt mode is selected, the user application must call `HAL_PPP_FUNCTION_IRQHandler()` (for example `HAL_PWR_PVD_IRQHandler()`), from `stm32l1xx_it.c` file, and implement `HAL_PPP_FUNCTIONCallback()` callback function (for example `HAL_PWR_PVDCallback()`).

2.11.6 DMA

The DMA HAL driver allows enabling and configuring the peripheral to be connected to the DMA Channels (except for internal SRAM/FLASH memory which do not require any initialization). Refer to the product reference manual for details on the DMA request corresponding to each peripheral.

For a given channel, `HAL_DMA_Init()` API allows programming the required configuration through the following parameters:

- Transfer Direction
- Source and Destination data formats
- Circular, Normal or peripheral flow control mode
- Channels Priority level
- Source and Destination Increment mode

Two operating modes are available:

- Polling mode I/O operation
 - a. Use `HAL_DMA_Start()` to start DMA transfer when the source and destination addresses and the Length of data to be transferred have been configured.
 - b. Use `HAL_DMA_PollForTransfer()` to poll for the end of current transfer. In this case a fixed timeout can be configured depending on the user application.
- Interrupt mode I/O operation
 - a. Configure the DMA interrupt priority using `HAL_NVIC_SetPriority()`
 - b. Enable the DMA IRQ handler using `HAL_NVIC_EnableIRQ()`
 - c. Use `HAL_DMA_Start_IT()` to start DMA transfer when the source and destination addresses and the length of data to be transferred have been configured. In this case the DMA interrupt is configured.
 - d. Use `HAL_DMA_IRQHandler()` called under `DMA_IRQHandler()` Interrupt subroutine
 - e. When data transfer is complete, `HAL_DMA_IRQHandler()` function is executed and a user function can be called by customizing `XferCpltCallback` and `XferErrorCallback` function pointer (i.e. a member of DMA handle structure).

Additional functions and macros are available to ensure efficient DMA management:

- Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
- Use HAL_DMA_Abort() function to abort the current transfer

The most used DMA HAL driver macros are the following:

- __HAL_DMA_ENABLE: enable the specified DMA Channels.
- __HAL_DMA_DISABLE: disables the specified DMA Channels.
- __HAL_DMA_GET_FLAG: gets the DMA Channels pending flags.
- __HAL_DMA_CLEAR_FLAG: clears the DMA Channels pending flags.
- __HAL_DMA_ENABLE_IT: enables the specified DMA Channels interrupts.
- __HAL_DMA_DISABLE_IT: disables the specified DMA Channels interrupts.
- __HAL_DMA_GET_IT_SOURCE: checks whether the specified DMA stream interrupt has occurred or not.



When a peripheral is used in DMA mode, the DMA initialization should be done in the HAL_PPP_MspInit() callback. In addition, the user application should associate the DMA handle to the PPP handle (refer to section “HAL IO operation functions”).



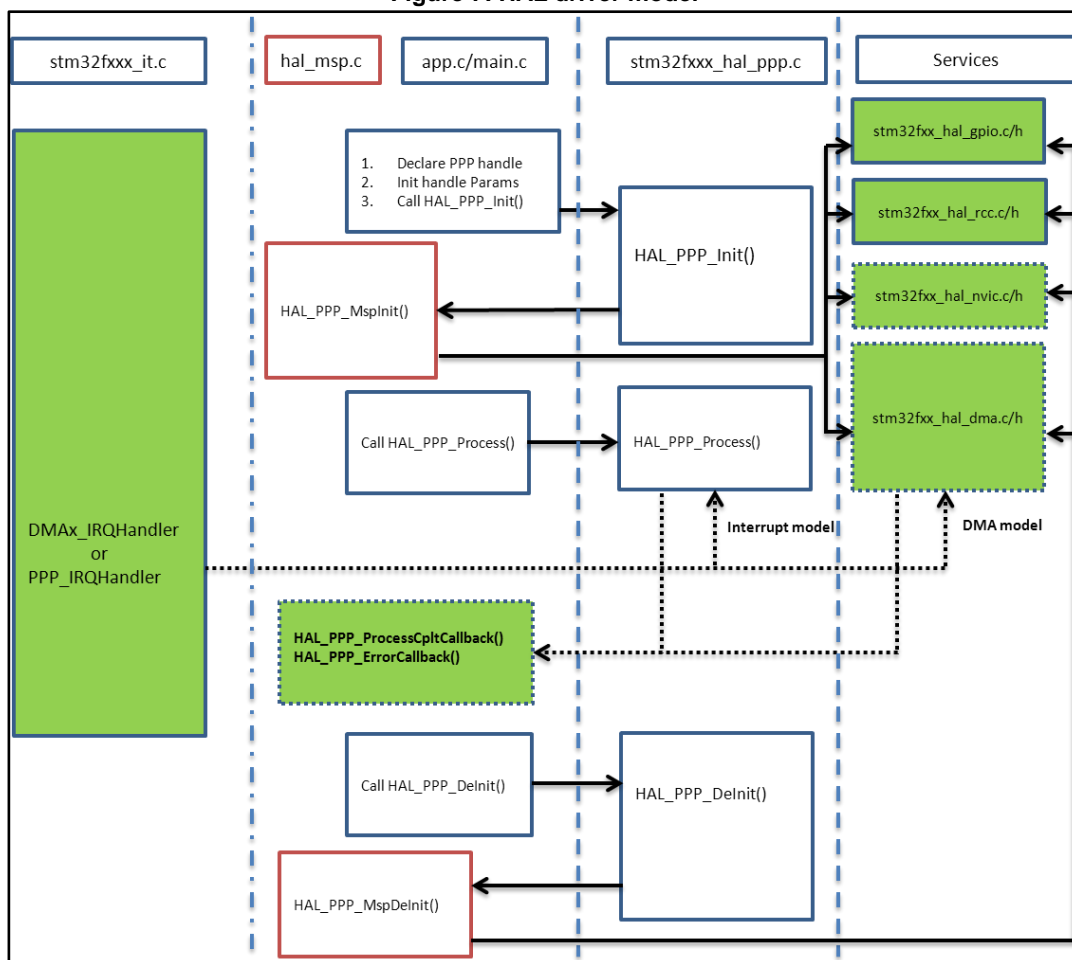
DMA channel callbacks need to be initialized by the user application only in case of memory-to-memory transfer. However when peripheral-to-memory transfers are used, these callbacks are automatically initialized by calling a process API function that uses the DMA.

2.12 How to use HAL drivers

2.12.1 HAL usage models

The following figure shows the typical use of the HAL driver and the interaction between the application user, the HAL driver and the interrupts.

Figure 7: HAL driver model



Basically, the HAL driver APIs are called from user files and optionally from interrupt handlers file when the APIs based on the DMA or the PPP peripheral dedicated interrupts are used.

When DMA or PPP peripheral interrupts are used, the PPP process complete callbacks are called to inform the user about the process completion in real-time event mode (interrupts). Note that the same process completion callbacks are used for DMA in interrupt mode.

2.12.2 HAL initialization

2.12.2.1 HAL global initialization

In addition to the peripheral initialization and de-initialization functions, a set of APIs are provided to initialize the HAL core implemented in file `stm32l1xx_hal.c`.

- **HAL_Init()**: this function must be called at application startup to
 - Initialize data/instruction cache and pre-fetch queue
 - Set SysTick timer to generate an interrupt each 1ms (based on HSI clock) with the lowest priority
 - Call **HAL_MspInit()** user callback function to perform system level initializations (Clock, GPIOs, DMA, interrupts). **HAL_MspInit()** is defined as “weak” empty function in the HAL drivers.
- **HAL_DeInit()**
 - Resets all peripherals

- Calls function `HAL_MspDeInit()` which is a user callback function to do system level De-Initializations.
- `HAL_GetTick()`: this function gets current SysTick counter value (incremented in SysTick interrupt) used by peripherals drivers to handle timeouts.
- `HAL_Delay()`: this function implements a delay (expressed in milliseconds) using the SysTick timer.
Care must be taken when using `HAL_Delay()` since this function provides an accurate delay (expressed in milliseconds) based on a variable incremented in SysTick ISR. This means that if `HAL_Delay()` is called from a peripheral ISR, then the SysTick interrupt must have highest priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR will be blocked.

2.12.2.2 System clock initialization

The clock configuration is done at the beginning of the user code. However the user can change the configuration of the clock in his own code. Please find below the typical Clock configuration sequence:

```
static void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;
  /* Enable MSI with range 5, PLL is OFF */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIRange = RCC_MSIRANGE_5;
  RCC_OscInitStruct.MSICalibrationValue=0x00;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  HAL_RCC_OscConfig(&RCC_OscInitStruct);
  /* Select MSI as system clock source and configure the HCLK, PCLK1 and PCLK2 clocks
  dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
  RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0);
}
```

2.12.2.3 HAL MSP initialization process

The peripheral initialization is done through `HAL_PPP_Init()` while the hardware resources initialization used by a peripheral (PPP) is performed during this initialization by calling MSP callback function `HAL_PPP_MspInit()`.

The `MspInit` callback performs the low level initialization related to the different additional hardware resources: RCC, GPIO, NVIC and DMA.

All the HAL drivers with handles include two MSP callbacks for initialization and de-initialization:

```
/**
 * @brief Initializes the PPP MSP.
 * @param hppp: PPP handle
 * @retval None */
void __weak HAL_PPP_MspInit(PPP_HandleTypeDef *hppp) {
  /* NOTE : This function Should not be modified, when the callback is needed,
  the HAL_PPP_MspInit could be implemented in the user file */
}
/**
 * @brief DeInitializes PPP MSP.
 * @param hppp: PPP handle
 * @retval None */
void __weak HAL_PPP_MspDeInit(PPP_HandleTypeDef *hppp) {
  /* NOTE : This function Should not be modified, when the callback is needed,
```

```
the HAL_PPP_MspDeInit could be implemented in the user file */
}
```

The MSP callbacks are declared empty as weak functions in each peripheral driver. The user can use them to set the low level initialization code or omit them and use his own initialization routine.

The HAL MSP callback is implemented inside the *stm32l1xx_hal_msp.c* file in the user folders. An *stm32l1xx_hal_msp.c* file template is located in the HAL folder and should be copied to the user folder. It can be generated automatically by STM32CubeMX tool and further modified. Note that all the routines are declared as weak functions and could be overwritten or removed to use user low level initialization code.

stm32l1xx_hal_msp.c file contains the following functions:

Table 14: MSP functions

Routine	Description
void HAL_MspInit()	Global MSP initialization routine
void HAL_MspDeInit()	Global MSP de-initialization routine
void HAL_PPP_MspInit()	PPP MSP initialization routine
void HAL_PPP_MspDeInit()	PPP MSP de-initialization routine

By default, if no peripheral needs to be de-initialized during the program execution, the whole MSP initialization is done in *Hal_MspInit()* and MSP De-Initialization in the *Hal_MspDeInit()*. In this case the *HAL_PPP_MspInit()* and *HAL_PPP_MspDeInit()* are not implemented.

When one or more peripherals needs to be de-initialized in run time and the low level resources of a given peripheral need to be released and used by another peripheral, *HAL_PPP_MspDeInit()* and *HAL_PPP_MspInit()* are implemented for the concerned peripheral and other peripherals initialization and de-Initialization are kept in the global *HAL_MspInit()* and the *HAL_MspDeInit()*.

If there is nothing to be initialized by the global *HAL_MspInit()* and *HAL_MspDeInit()*, the two routines can simply be omitted.

2.12.3 HAL IO operation process

The HAL functions with internal data processing like Transmit, Receive, Write and Read are generally provided with three data processing modes as follows:

- Polling mode
- Interrupt mode
- DMA mode

2.12.3.1 Polling mode

In polling mode, the HAL functions return the process status when the data processing in blocking mode is complete. The operation is considered complete when the function returns the HAL_OK status, otherwise an error status is returned. The user can get more information through the *HAL_PPP_GetState()* function. The data processing is handled internally in a loop. A timeout (expressed in ms) is used to prevent process hanging.

The example below shows the typical polling mode processing sequence :

```
HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t pData,
int16_t tSize, uint32_t tTimeout)
{
```



```

if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(...) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
}
(...)
return HELIAC; }

```

2.12.3.2 Interrupt mode

In Interrupt mode, the HAL function returns the process status after starting the data processing and enabling the appropriate interruption. The end of the operation is indicated by a callback declared as a weak function. It can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function.

In interrupt mode, four functions are declared in the driver:

- *HAL_PPP_Process_IT()*: launch the process
- *HAL_PPP_IRQHandler()*: the global PPP peripheral interruption
- *__weak HAL_PPP_ProcessCpltCallback ()*: the callback relative to the process completion.
- *__weak HAL_PPP_ProcessErrorCallback()*: the callback relative to the process Error.

To use a process in interrupt mode, *HAL_PPP_Process_IT()* is called in the user file and *HAL_PPP_IRQHandler* in *stm32l1xx_it.c*.

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver. This means that the user can declare it again in the application. The function in the driver is not modified.

An example of use is illustrated below:

main.c file:

```

UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = USART1;
HAL_UART_Init(&UartHandle);
HAL_UART_SendIT(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
}

```

stm32l1xx_it.c file:

```

extern UART_HandleTypeDef UartHandle;

```

```
void USART1_IRQHandler(void)
{
  HAL_UART_IRQHandler(&UartHandle);
}
```

2.12.3.3 DMA mode

In DMA mode, the HAL function returns the process status after starting the data processing through the DMA and after enabling the appropriate DMA interruption. The end of the operation is indicated by a callback declared as a weak function and can be customized by the user to be informed in real-time about the process completion. The user can also get the process status through the *HAL_PPP_GetState()* function. For the DMA mode, three functions are declared in the driver:

- *HAL_PPP_Process_DMA()*: launch the process
- *HAL_PPP_DMA_IRQHandler()*: the DMA interruption used by the PPP peripheral
- *__weak HAL_PPP_ProcessCpltCallback()*: the callback relative to the process completion.
- *__weak HAL_PPP_ErrorCpltCallback()*: the callback relative to the process Error.

To use a process in DMA mode, *HAL_PPP_Process_DMA()* is called in the user file and the *HAL_PPP_DMA_IRQHandler()* is placed in the *stm32l1xx_it.c*. When DMA mode is used, the DMA initialization is done in the *HAL_PPP_MspInit()* callback. The user should also associate the DMA handle to the PPP handle. For this purpose, the handles of all the peripheral drivers that use the DMA must be declared as follows:

```
typedef struct
{
  PPP_TypeDef *Instance; /* Register base address */
  PPP_InitTypeDef Init; /* PPP communication parameters */
  HAL_StateTypeDef State; /* PPP communication state */
  (...)
  DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

The initialization is done as follows (UART example):

```
int main(void)
{
  /* Set User Parameters */
  UartHandle.Init.BaudRate = 9600;
  UartHandle.Init.WordLength = UART_DATABITS_8;
  UartHandle.Init.StopBits = UART_STOPBITS_1;
  UartHandle.Init.Parity = UART_PARITY_NONE;
  UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  UartHandle.Init.Mode = UART_MODE_TX_RX;
  UartHandle.Init.Instance = USART1;
  HAL_UART_Init(&UartHandle);
  (...)
}

void HAL_UART_MspInit (UART_HandleTypeDef * huart)
{
  static DMA_HandleTypeDef hdma_tx;
  static DMA_HandleTypeDef hdma_rx;
  (...)
  HAL_LINKDMA(UartHandle, DMA_Handle tx, hdma_tx);
  HAL_LINKDMA(UartHandle, DMA_Handle rx, hdma_rx);
  (...)
}
```

The *HAL_PPP_ProcessCpltCallback()* function is declared as weak function in the driver that means, the user can declare it again in the application code. The function in the driver should not be modified.

An example of use is illustrated below:

main.c file:

```

UART_HandleTypeDef UartHandle;
int main(void)
{
  /* Set User Paramaters */
  UartHandle.Init.BaudRate = 9600;
  UartHandle.Init.WordLength = UART_DATABITS_8;
  UartHandle.Init.StopBits = UART_STOPBITS_1;
  UartHandle.Init.Parity = UART_PARITY_NONE;
  UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  UartHandle.Init.Mode = UART_MODE_TX_RX; UartHandle.Init.Instance = USART1;
  HAL_UART_Init(&UartHandle);
  HAL_UART_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
  while (1);
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
}

void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
}

```

stm32l1xx_it.c file:

```

extern UART_HandleTypeDef UartHandle;
void DMAx_IRQHandler(void)
{
  HAL_DMA_IRQHandler(&UartHandle.DMA_Handle_tx);
}

```

HAL_USART_TxCpltCallback() and *HAL_USART_ErrorCallback()* should be linked in the *HAL_PPP_Process_DMA()* function to the DMA transfer complete callback and the DMA transfer Error callback by using the following statement:

```

HAL_PPP_Process_DMA (PPP_HandleTypeDef *hppp, Params...)
{
  (...)
  hppp->DMA_Handle->XferCpltCallback = HAL_UART_TxCpltCallback ;
  hppp->DMA_Handle->XferErrorCallback = HAL_UART_ErrorCallback ;
  (...)
}

```

2.12.4 Timeout and error management

2.12.4.1 Timeout management

The timeout is often used for the APIs that operate in polling mode. It defines the delay during which a blocking process should wait till an error is returned. An example is provided below:

```

HAL_StatusTypeDef HAL_DMA_PollForTransfer(DMA_HandleTypeDef *hdma, uint32_t
CompleteLevel, uint32_t Timeout)

```

The timeout possible value are the following:

Table 15: Timeout values

Timeout value	Description
0	No poll : Immediate process check and exit
1 ... (HAL_MAX_DELAY -1) ⁽¹⁾	Timeout in ms
HAL_MAX_DELAY	Infinite poll till process is successful

Notes:

⁽¹⁾HAL_MAX_DELAY is defined in the stm32l1xx_hal_def.h as 0xFFFFFFFF

However, in some cases, a fixed timeout is used for system peripherals or internal HAL driver processes. In these cases, the timeout has the same meaning and is used in the same way, except when it is defined locally in the drivers and cannot be modified or introduced as an argument in the user application.

Example of fixed timeout:

```
#define LOCAL_PROCESS_TIMEOUT 100
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef)
{
    (...)
    timeout = HAL_GetTick() + LOCAL_PROCESS_TIMEOUT;
    (...)
    while(ProcessOngoing)
    {
        (...)
        if(HAL_GetTick() >= timeout)
        {
            /* Process unlocked */
            HAL_UNLOCK(hppp);
            hppp->State= HAL_PPP_STATE_TIMEOUT;
            return HAL_PPP_STATE_TIMEOUT;
        }
        (...)
    }
}
```

The following example shows how to use the timeout inside the polling functions:

```
HAL_PPP_StateTypeDef HAL_PPP_Poll (PPP_HandleTypeDef *hppp, uint32_t Timeout)
{
    (...)
    timeout = HAL_GetTick() + Timeout;
    (...)
    while(ProcessOngoing)
    {
        (...)
        if(Timeout != HAL_MAX_DELAY)
        {
            if(HAL_GetTick() >= timeout)
            {
                /* Process unlocked */
                HAL_UNLOCK(hppp);
                hppp->State= HAL_PPP_STATE_TIMEOUT;
                return hppp->State;
            }
        }
        (...)
    }
}
```

2.12.4.2 Error management

The HAL drivers implement a check for the following items:

- Valid parameters: for some process the used parameters should be valid and already defined, otherwise the system can crash or go into an undefined state. These critical parameters are checked before they are used (see example below).

```
HAL_StatusTypeDef HAL_PPP_Process(PPP_HandleTypeDef* hppp, uint32_t *pdata, uint32_t Size)
{
    if ((pdata == NULL) || (Size == 0))
    {
        return HAL_ERROR;
    }
}
```

- Valid handle: the PPP peripheral handle is the most important argument since it keeps the PPP driver vital parameters. It is always checked in the beginning of the *HAL_PPP_Init()* function.

```
HAL StatusTypeDef HAL_PPP_Init(PPP_HandleTypeDef* hppp)
{
  if (hppp == NULL) //the handle should be already allocated
  {
    return HAL_ERROR;
  }
}
```

- Timeout error: the following statement is used when a timeout error occurs: while (Process ongoing)

```
{
  timeout = HAL_GetTick() + Timeout; while (data processing is running)
  {
    if(timeout) { return HAL_TIMEOUT;
  }
}
```

When an error occurs during a peripheral process, *HAL_PPP_Process()* returns with a *HAL_ERROR* status. The HAL PPP driver implements the *HAL_PPP_GetError()* to allow retrieving the origin of the error.

```
HAL_PPP_ErrorTypeDef HAL_PPP_GetError (PPP_HandleTypeDef *hppp);
```

In all peripheral handles, a *HAL_PPP_ErrorTypeDef* is defined and used to store the last error code.

```
typedef struct
{
  PPP_TypeDef * Instance; /* PPP registers base address */
  PPP_InitTypeDef Init; /* PPP initialization parameters */
  HAL_LockTypeDef Lock; /* PPP locking object */
  IO HAL_PPP_StateTypeDef State; /* PPP state */
  IO HAL_PPP_ErrorTypeDef ErrorCode; /* PPP Error code */
  (...)
  /* PPP specific parameters */
}
PPP_HandleTypeDef;
```

The error state and the peripheral global state are always updated before returning an error:

```
PPP->State = HAL_PPP_READY; /* Set the peripheral ready */
PPP->ErrorCode = HAL_ERRORCODE ; /* Set the error code */
HAL_UNLOCK(PPP) ; /* Unlock the PPP resources */
return HAL_ERROR; /*return with HAL error */
```

HAL_PPP_GetError () must be used in interrupt mode in the error callback:

```
void HAL_PPP_ProcessCpltCallback(PPP_HandleTypeDef *hspi)
{
  ErrorCode = HAL_PPP_GetError (hspi); /* retrieve error code */
}
```

2.12.4.3 Run-time checking

The HAL implements run-time failure detection by checking the input values of all HAL drivers functions. The run-time checking is achieved by using an **assert_param** macro. This macro is used in all the HAL drivers' functions which have an input parameter. It allows verifying that the input value lies within the parameter allowed values.

To enable the run-time checking, use the **assert_param** macro, and leave the define **USE_FULL_ASSERT** uncommented in **stm32l1xx_hal_conf.h** file.

```
void HAL_UART_Init(UART_HandleTypeDef *huart)
{
  (..) /* Check the parameters */
  assert_param(IS_UART_INSTANCE(huart->Instance));
  assert_param(IS_UART_BAUDRATE(huart->Init.BaudRate));
  assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));
  assert_param(IS_UART_STOPBITS(huart->Init.StopBits));
  assert_param(IS_UART_PARITY(huart->Init.Parity));
  assert_param(IS_UART_MODE(huart->Init.Mode));
  assert_param(IS_UART_HARDWARE_FLOW_CONTROL(huart->Init.HwFlowCtl));
  (..)

  /** @defgroup UART Word Length *
  @{
  */
  #define UART_WORDLENGTH_8B ((uint32_t)0x00000000)
  #define UART_WORDLENGTH_9B ((uint32_t)USART_CR1_M)
  #define IS_UART_WORD_LENGTH(LENGTH) (((LENGTH) == UART_WORDLENGTH_8B) ||
  \ ((LENGTH) == UART_WORDLENGTH_9B))
```

If the expression passed to the **assert_param** macro is false, the **assert_failed** function is called and returns the name of the source file and the source line number of the call that failed. If the expression is true, no value is returned.

The **assert_param** macro is implemented in **stm32l1xx_hal_conf.h**:

```
/* Exported macro -----*/
#ifdef USE_FULL_ASSERT
/**
 * @brief The assert_param macro is used for function's parameters check.
 * @param expr: If expr is false, it calls assert_failed function
 * which reports the name of the source file and the source
 * line number of the call that failed.
 * If expr is true, it returns no value.
 * @retval None */
#define assert_param(expr) ((expr)?(void)0:assert_failed((uint8_t *) FILE ,
  LINE ))
/* Exported functions -----*/
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr) ((void)0)
#endif /* USE_FULL_ASSERT */
```

The **assert_failed** function is implemented in the **main.c** file or in any other user C file:

```
#ifdef USE_FULL_ASSERT /**
 * @brief Reports the name of the source file and the source line number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None */
void assert_failed(uint8_t* file, uint32_t line)
{
  (..)
}
```

```
/* User can add his own implementation to report the file name and line number,  
ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */  
/* Infinite loop */  
while (1)  
{  
}  
}
```



Because of the overhead run-time checking introduces, it is recommended to use it during application code development and debugging, and to remove it from the final application to improve code size and speed.

3 HAL System Driver

3.1 HAL System Driver

3.2 HAL Firmware driver API description

3.2.1 How to use this driver

The common HAL driver contains a set of generic and common APIs that can be used by the PPP peripheral drivers and the user to start using the HAL.

The HAL contains two APIs' categories:

- Common HAL APIs
- Services HAL APIs

3.2.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initializes the Flash interface, the NVIC allocation and initial clock configuration. It initializes the source of time base also when timeout is needed and the backup domain when enabled.
- de-Initializes common part of the HAL.
- Configure The time base source to have 1ms time base with a dedicated Tick interrupt priority.
 - SysTick timer is used by default as source of time base, but user can eventually implement his proper time base source (a general purpose timer for example or other time source), keeping in mind that Time base duration should be kept 1ms since PPP_TIMEOUT_VALUES are defined and handled in milliseconds basis.
 - Time base configuration function (HAL_InitTick ()) is called automatically at the beginning of the program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().
 - Source of time base is configured to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, the Tick interrupt line must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked.
 - functions affecting time base configurations are declared as __Weak to make override possible in case of other implementations in user file.

This section contains the following APIs:

- [*HAL_Init\(\)*](#)
- [*HAL_DeInit\(\)*](#)
- [*HAL_MspInit\(\)*](#)
- [*HAL_MspDeInit\(\)*](#)
- [*HAL_InitTick\(\)*](#)

3.2.3 HAL Control functions

This section provides functions allowing to:

- Provide a tick value in millisecond
- Provide a blocking delay in millisecond

- Suspend the time base source interrupt
- Resume the time base source interrupt
- Get the HAL API driver version
- Get the device identifier
- Get the device revision identifier
- Enable/Disable Debug module during Sleep mode
- Enable/Disable Debug module during STOP mode
- Enable/Disable Debug module during STANDBY mode

This section contains the following APIs:

- [*HAL_IncTick\(\)*](#)
- [*HAL_GetTick\(\)*](#)
- [*HAL_Delay\(\)*](#)
- [*HAL_SuspendTick\(\)*](#)
- [*HAL_ResumeTick\(\)*](#)
- [*HAL_GetHalVersion\(\)*](#)
- [*HAL_GetREVID\(\)*](#)
- [*HAL_GetDEVID\(\)*](#)
- [*HAL_DBGMCU_EnableDBGSleepMode\(\)*](#)
- [*HAL_DBGMCU_DisableDBGSleepMode\(\)*](#)
- [*HAL_DBGMCU_EnableDBGStopMode\(\)*](#)
- [*HAL_DBGMCU_DisableDBGStopMode\(\)*](#)
- [*HAL_DBGMCU_EnableDBGStandbyMode\(\)*](#)
- [*HAL_DBGMCU_DisableDBGStandbyMode\(\)*](#)

3.2.4 HAL_Init

Function Name	HAL_StatusTypeDef HAL_Init (void)
Function Description	This function configures the Flash prefetch, Configures time base source, NVIC and Low level hardware.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is called at the beginning of program after reset and before the clock configuration • The time base configuration is based on MSI clock when exiting from Reset. Once done, time base tick start incrementing. In the default implementation, SysTick is used as source of time base. the tick variable is incremented each 1ms in its ISR.

3.2.5 HAL_DeInit

Function Name	HAL_StatusTypeDef HAL_DeInit (void)
Function Description	This function de-Initializes common part of the HAL and stops the source of time base.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is optional.

3.2.6 HAL_MspInit

Function Name	void HAL_MspInit (void)
---------------	---------------------------------

	Function Description	Initializes the MSP.
	Return values	<ul style="list-style-type: none"> None
3.2.7	HAL_MspDeInit	
	Function Name	void HAL_MspDeInit (void)
	Function Description	DeInitializes the MSP.
	Return values	<ul style="list-style-type: none"> None
3.2.8	HAL_InitTick	
	Function Name	HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)
	Function Description	This function configures the source of the time base.
	Parameters	<ul style="list-style-type: none"> TickPriority: Tick interrupt priority.
	Return values	<ul style="list-style-type: none"> HAL status
	Notes	<ul style="list-style-type: none"> This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is reconfigured by HAL_RCC_ClockConfig(). In the default implementation, SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Care must be taken if HAL_Delay() is called from a peripheral ISR process, The the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise the caller ISR process will be blocked. The function is declared as __Weak to be overwritten in case of other implementation in user file.
3.2.9	HAL_IncTick	
	Function Name	void HAL_IncTick (void)
	Function Description	This function is called to increment a global variable "uwTick" used as application time base.
	Return values	<ul style="list-style-type: none"> None
	Notes	<ul style="list-style-type: none"> In the default implementation, this variable is incremented each 1ms in SysTick ISR. This function is declared as __weak to be overwritten in case of other implementations in user file.
3.2.10	HAL_GetTick	
	Function Name	uint32_t HAL_GetTick (void)
	Function Description	Provides a tick value in millisecond.
	Return values	<ul style="list-style-type: none"> tick value
	Notes	<ul style="list-style-type: none"> This function is declared as __weak to be overwritten in case of other implementations in user file.
3.2.11	HAL_Delay	

Function Name	void HAL_Delay (__IO uint32_t Delay)
Function Description	This function provides accurate delay (in milliseconds) based on variable incremented.
Parameters	<ul style="list-style-type: none"> Delay: specifies the delay time length, in milliseconds.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals where uwTick is incremented. This function is declared as <code>__weak</code> to be overwritten in case of other implementations in user file.

3.2.12 HAL_SuspendTick

Function Name	void HAL_SuspendTick (void)
Function Description	Suspend Tick increment.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_SuspendTick() is called, the the SysTick interrupt will be disabled and so Tick increment is suspended. This function is declared as <code>__weak</code> to be overwritten in case of other implementations in user file.

3.2.13 HAL_ResumeTick

Function Name	void HAL_ResumeTick (void)
Function Description	Resume Tick increment.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> In the default implementation , SysTick timer is the source of time base. It is used to generate interrupts at regular time intervals. Once HAL_ResumeTick() is called, the the SysTick interrupt will be enabled and so Tick increment is resumed. This function is declared as <code>__weak</code> to be overwritten in case of other implementations in user file.

3.2.14 HAL_GetHalVersion

Function Name	uint32_t HAL_GetHalVersion (void)
Function Description	Returns the HAL revision.
Return values	<ul style="list-style-type: none"> version 0xXYZR (8bits for each decimal, R for RC)

3.2.15 HAL_GetREVID

Function Name	uint32_t HAL_GetREVID (void)
Function Description	Returns the device revision identifier.
Return values	<ul style="list-style-type: none"> Device revision identifier

3.2.16 HAL_GetDEVID

Function Name **uint32_t HAL_GetDEVID (void)**

Function Description Returns the device identifier.

Return values • Device identifier

3.2.17 HAL_DBGMCU_EnableDBGSleepMode

Function Name **void HAL_DBGMCU_EnableDBGSleepMode (void)**

Function Description Enable the Debug Module during SLEEP mode.

Return values • None

3.2.18 HAL_DBGMCU_DisableDBGSleepMode

Function Name **void HAL_DBGMCU_DisableDBGSleepMode (void)**

Function Description Disable the Debug Module during SLEEP mode.

Return values • None

3.2.19 HAL_DBGMCU_EnableDBGStopMode

Function Name **void HAL_DBGMCU_EnableDBGStopMode (void)**

Function Description Enable the Debug Module during STOP mode.

Return values • None

3.2.20 HAL_DBGMCU_DisableDBGStopMode

Function Name **void HAL_DBGMCU_DisableDBGStopMode (void)**

Function Description Disable the Debug Module during STOP mode.

Return values • None

3.2.21 HAL_DBGMCU_EnableDBGStandbyMode

Function Name **void HAL_DBGMCU_EnableDBGStandbyMode (void)**

Function Description Enable the Debug Module during STANDBY mode.

Return values • None

3.2.22 HAL_DBGMCU_DisableDBGStandbyMode

Function Name **void HAL_DBGMCU_DisableDBGStandbyMode (void)**

Function Description Disable the Debug Module during STANDBY mode.

Return values • None

3.3 HAL Firmware driver defines**3.3.1 HAL**

HAL Private Defines

__STM32L1xx_HAL_VERSION_MAIN [31:24] main version
__STM32L1xx_HAL_VERSION_SUB1 [23:16] sub1 version
__STM32L1xx_HAL_VERSION_SUB2 [15:8] sub2 version
__STM32L1xx_HAL_VERSION_RC [7:0] release candidate
__STM32L1xx_HAL_VERSION
IDCODE_DEVID_MASK

4 HAL ADC Generic Driver

4.1 HAL ADC Generic Driver

4.2 ADC Firmware driver registers structures

4.2.1 ADC_InitTypeDef

Data Fields

- *uint32_t* **ClockPrescaler**
- *uint32_t* **Resolution**
- *uint32_t* **DataAlign**
- *uint32_t* **ScanConvMode**
- *uint32_t* **EOCSelection**
- *uint32_t* **LowPowerAutoWait**
- *uint32_t* **LowPowerAutoPowerOff**
- *uint32_t* **ChannelsBank**
- *uint32_t* **ContinuousConvMode**
- *uint32_t* **NbrOfConversion**
- *uint32_t* **DiscontinuousConvMode**
- *uint32_t* **NbrOfDiscConversion**
- *uint32_t* **ExternalTrigConv**
- *uint32_t* **ExternalTrigConvEdge**
- *uint32_t* **DMAContinuousRequests**

Field Documentation

- *uint32_t* **ADC_InitTypeDef::ClockPrescaler**
Select ADC clock source (asynchronous clock derived from HSI RC oscillator) and clock prescaler. This parameter can be a value of [ADC_ClockPrescaler](#) Note: In case of usage of channels on injected group, ADC frequency should be lower than AHB clock frequency /4 for resolution 12 or 10 bits, AHB clock frequency /3 for resolution 8 bits, AHB clock frequency /2 for resolution 6 bits. Note: HSI RC oscillator must be preliminarily enabled at RCC top level.
- *uint32_t* **ADC_InitTypeDef::Resolution**
Configures the ADC resolution. This parameter can be a value of [ADC_Resolution](#)
- *uint32_t* **ADC_InitTypeDef::DataAlign**
Specifies ADC data alignment to right (MSB on register bit 11 and LSB on register bit 0) (default setting) or to left (if regular group: MSB on register bit 15 and LSB on register bit 4, if injected group (MSB kept as signed value due to potential negative value after offset application): MSB on register bit 14 and LSB on register bit 3). This parameter can be a value of [ADC_Data_align](#)
- *uint32_t* **ADC_InitTypeDef::ScanConvMode**
Configures the sequencer of regular and injected groups. This parameter can be associated to parameter 'DiscontinuousConvMode' to have main sequence subdivided in successive parts. If disabled: Conversion is performed in single mode (one channel converted, the one defined in rank 1). Parameters 'NbrOfConversion' and 'InjectedNbrOfConversion' are discarded (equivalent to set to 1). If enabled: Conversions are performed in sequence mode (multiple ranks defined by

'NbrOfConversion'/'InjectedNbrOfConversion' and each channel rank). Scan direction is upward: from rank1 to rank 'n'. This parameter can be a value of [ADC_Scan_mode](#)

- ***uint32_t ADC_InitTypeDef::EOCSelection***
Specifies what EOC (End Of Conversion) flag is used for conversion by polling and interruption: end of conversion of each rank or complete sequence. This parameter can be a value of [ADC_EOCSelection](#). Note: For injected group, end of conversion (flag&IT) is raised only at the end of the sequence. Therefore, if end of conversion is set to end of each conversion, injected group should not be used with interruption (HAL_ADCEX_InjectedStart_IT) or polling (HAL_ADCEX_InjectedStart and HAL_ADCEX_InjectedPollForConversion). By the way, polling is still possible since driver will use an estimated timing for end of injected conversion. Note: If overrun feature is intended to be used, use ADC in mode 'interruption' (function **HAL_ADC_Start_IT()**) with parameter EOCSelection set to end of each conversion or in mode 'transfer by DMA' (function **HAL_ADC_Start_DMA()**). If overrun feature is intended to be bypassed, use ADC in mode 'polling' or 'interruption' with parameter EOCSelection must be set to end of sequence
- ***uint32_t ADC_InitTypeDef::LowPowerAutoWait***
Selects the dynamic low power Auto Delay: new conversion start only when the previous conversion (for regular group) or previous sequence (for injected group) has been treated by user software, using function **HAL_ADC_GetValue()** or **HAL_ADCEX_InjectedGetValue()**. This feature automatically adapts the speed of ADC to the speed of the system that reads the data. Moreover, this avoids risk of overrun for low frequency applications. This parameter can be a value of [ADC_LowPowerAutoWait](#). Note: Do not use with interruption or DMA (**HAL_ADC_Start_IT()**, **HAL_ADC_Start_DMA()**) since they have to clear immediately the EOC flag to free the IRQ vector sequencer. Do use with polling: 1. Start conversion with **HAL_ADC_Start()**, 2. Later on, when conversion data is needed: use **HAL_ADC_PollForConversion()** to ensure that conversion is completed and use **HAL_ADC_GetValue()** to retrieve conversion result and trig another conversion (in case of usage of injected group, use the equivalent functions **HAL_ADCEX_Injected_Start()**, **HAL_ADCEX_InjectedGetValue()**, ...). Note: ADC clock latency and some timing constraints depending on clock prescaler have to be taken into account: refer to reference manual (register ADC_CR2 bit DELS description).
- ***uint32_t ADC_InitTypeDef::LowPowerAutoPowerOff***
Selects the auto-off mode: the ADC automatically powers-off after a conversion and automatically wakes-up when a new conversion is triggered (with startup time between trigger and start of sampling). This feature can be combined with automatic wait mode (parameter 'LowPowerAutoWait'). This parameter can be a value of [ADC_LowPowerAutoPowerOff](#).
- ***uint32_t ADC_InitTypeDef::ChannelsBank***
Selects the ADC channels bank. This parameter can be a value of [ADC_ChannelsBank](#). Note: Banks availability depends on devices categories. Note: To change bank selection on the fly, without going through execution of **HAL_ADC_Init()**, macro '**__HAL_ADC_CHANNELS_BANK()**' can be used directly.
- ***uint32_t ADC_InitTypeDef::ContinuousConvMode***
Specifies whether the conversion is performed in single mode (one conversion) or continuous mode for regular group, after the selected trigger occurred (software start or external trigger). This parameter can be set to ENABLE or DISABLE.
- ***uint32_t ADC_InitTypeDef::NbrOfConversion***
Specifies the number of ranks that will be converted within the regular group sequencer. To use regular group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 28.
- ***uint32_t ADC_InitTypeDef::DiscontinuousConvMode***
Specifies whether the conversions sequence of regular group is performed in

Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE.

- ***uint32_t ADC_InitTypeDef::NbrOfDiscConversion***
Specifies the number of discontinuous conversions in which the main sequence of regular group (parameter NbrOfConversion) will be subdivided. If parameter 'DiscontinuousConvMode' is disabled, this parameter is discarded. This parameter must be a number between Min_Data = 1 and Max_Data = 8.
- ***uint32_t ADC_InitTypeDef::ExternalTrigConv***
Selects the external event used to trigger the conversion start of regular group. If set to ADC_SOFTWARE_START, external triggers are disabled. If set to external trigger source, triggering is on event rising edge. This parameter can be a value of [ADC_External_trigger_source_Regular](#)
- ***uint32_t ADC_InitTypeDef::ExternalTrigConvEdge***
Selects the external trigger edge of regular group. If trigger is set to ADC_SOFTWARE_START, this parameter is discarded. This parameter can be a value of [ADC_External_trigger_edge_Regular](#)
- ***uint32_t ADC_InitTypeDef::DMAContinuousRequests***
Specifies whether the DMA requests are performed in one shot mode (DMA transfer stop when number of conversions is reached) or in Continuous mode (DMA transfer unlimited, whatever number of conversions). Note: In continuous mode, DMA must be configured in circular mode. Otherwise an overrun will be triggered when DMA buffer maximum pointer is reached. This parameter can be set to ENABLE or DISABLE. Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled without continuous mode or external trigger that could launch a conversion).

4.2.2 ADC_ChannelConfTypeDef

Data Fields

- ***uint32_t Channel***
- ***uint32_t Rank***
- ***uint32_t SamplingTime***

Field Documentation

- ***uint32_t ADC_ChannelConfTypeDef::Channel***
Specifies the channel to configure into ADC regular group. This parameter can be a value of [ADC_channels](#) Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability. Maximum number of channels by device category (without taking in account each device package constraints): STM32L1 category 1, 2: 24 channels on external pins + 3 channels on internal measurement paths (VrefInt, Temp sensor, Vcomp): Channel 0 to channel 26. STM32L1 category 3: 25 channels on external pins + 3 channels on internal measurement paths (VrefInt, Temp sensor, Vcomp): Channel 0 to channel 26, 1 additional channel in bank B. Note: OPAMP1 and OPAMP2 are connected internally but not increasing internal channels number: they are sharing ADC input with external channels ADC_IN3 and ADC_IN8. STM32L1 category 4, 5: 40 channels on external

pins + 3 channels on internal measurement paths (VrefInt, Temp sensor, Vcomp): Channel 0 to channel 31, 11 additional channels in bank B. Note: OPAMP1 and OPAMP2 are connected internally but not increasing internal channels number: they are sharing ADC input with external channels ADC_IN3 and ADC_IN8. Note: In case of peripherals OPAMPx not used: 3 channels (3, 8, 13) can be configured as direct channels (fast channels). Refer to macro '**__HAL_ADC_CHANNEL_SPEED_FAST()**'. Note: In case of peripheral OPAMP3 and ADC channel OPAMP3 used (OPAMP3 available on STM32L1 devices Cat.4 only): the analog switch COMP1_SW1 must be closed. Refer to macro: '**__HAL_OPAMP_OPAMP3OUT_CONNECT_ADC_COMP1()**'.

- ***uint32_t ADC_ChannelConfTypeDef::Rank***
Specifies the rank in the regular group sequencer. This parameter can be a value of [ADC_regular_rank](#). Note: In case of need to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions can be adjusted).
- ***uint32_t ADC_ChannelConfTypeDef::SamplingTime***
Sampling time value to be set for the selected channel. Unit: ADC clock cycles. Conversion time is the addition of sampling time and processing time (12 ADC clock cycles at ADC resolution 12 bits, 11 cycles at 10 bits, 9 cycles at 8 bits, 7 cycles at 6 bits). This parameter can be a value of [ADC_sampling_times](#). Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting). Refer to device datasheet for timings values, parameters TS_vrefint, TS_temp (values rough order: 4us min).

4.2.3 ADC_AnalogWDGConfTypeDef

Data Fields

- ***uint32_t WatchdogMode***
- ***uint32_t Channel***
- ***uint32_t ITMode***
- ***uint32_t HighThreshold***
- ***uint32_t LowThreshold***
- ***uint32_t WatchdogNumber***

Field Documentation

- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogMode***
Configures the ADC analog watchdog mode: single/all channels, regular/injected group. This parameter can be a value of [ADC_analog_watchdog_mode](#).
- ***uint32_t ADC_AnalogWDGConfTypeDef::Channel***
Selects which ADC channel to monitor by analog watchdog. This parameter has an effect only if watchdog mode is configured on single channel (parameter WatchdogMode). This parameter can be a value of [ADC_channels](#).
- ***uint32_t ADC_AnalogWDGConfTypeDef::ITMode***
Specifies whether the analog watchdog is configured in interrupt or polling mode. This parameter can be set to ENABLE or DISABLE.

- ***uint32_t ADC_AnalogWDGConfTypeDef::HighThreshold***
Configures the ADC analog watchdog High threshold value. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF.
- ***uint32_t ADC_AnalogWDGConfTypeDef::LowThreshold***
Configures the ADC analog watchdog Low threshold value. This parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFFF.
- ***uint32_t ADC_AnalogWDGConfTypeDef::WatchdogNumber***
Reserved for future use, can be set to 0

4.2.4 ADC_HandleTypeDef

Data Fields

- ***ADC_TypeDef * Instance***
- ***ADC_InitTypeDef Init***
- ***__IO uint32_t NbrOfConversionRank***
- ***DMA_HandleTypeDef * DMA_Handle***
- ***HAL_LockTypeDef Lock***
- ***__IO uint32_t State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***ADC_TypeDef* ADC_HandleTypeDef::Instance***
Register base address
- ***ADC_InitTypeDef ADC_HandleTypeDef::Init***
ADC required parameters
- ***__IO uint32_t ADC_HandleTypeDef::NbrOfConversionRank***
ADC conversion rank counter
- ***DMA_HandleTypeDef* ADC_HandleTypeDef::DMA_Handle***
Pointer DMA Handler
- ***HAL_LockTypeDef ADC_HandleTypeDef::Lock***
ADC locking object
- ***__IO uint32_t ADC_HandleTypeDef::State***
ADC communication state (bitmap of ADC states)
- ***__IO uint32_t ADC_HandleTypeDef::ErrorCode***
ADC Error code

4.3 ADC Firmware driver API description

4.3.1 ADC peripheral features

- 12-bit, 10-bit, 8-bit or 6-bit configurable resolution
- Interrupt generation at the end of regular conversion, end of injected conversion, and in case of analog watchdog or overrun events.
- Single and continuous conversion modes.
- Scan mode for conversion of several channels sequentially.
- Data alignment with in-built data coherency.

- Programmable sampling time (channel wise)
- ADC conversion of regular group and injected group.
- External trigger (timer or EXTI) with configurable polarity for both regular and injected groups.
- DMA request generation for transfer of conversions data of regular group.
- ADC calibration
- ADC offset on injected channels
- ADC supply requirements: 2.4 V to 3.6 V at full speed and down to 1.8 V at slower speed.
- ADC input range: from Vref- (connected to Vssa) to Vref+ (connected to Vdda or to an external voltage reference).

4.3.2 How to use this driver

Configuration of top level parameters related to ADC

1. Enable the ADC interface
 - As prerequisite, ADC clock must be configured at RCC top level. Caution: On STM32L1, ADC clock frequency max is 16MHz (refer to device datasheet). Therefore, ADC clock prescaler must be configured in function of ADC clock source frequency to remain below this maximum frequency.
 - Two clock settings are mandatory:
 - ADC clock (core clock).
 - ADC clock (conversions clock). Only one possible clock source: derived from HSI RC 16MHz oscillator (HSI). ADC is connected directly to HSI RC 16MHz oscillator. Therefore, RCC PLL setting has no impact on ADC. PLL can be disabled ("PLL.PLLState = RCC_PLL_NONE") or enabled with HSI16 as clock source ("PLL.PLLSource = RCC_PLLSOURCE_HSI") to be used as device main clock source SYSCLK. The only mandatory setting is ".HSIState = RCC_HSI_ON"
 - Example: Into HAL_ADC_MspInit() (recommended code location) or with other device clock parameters configuration:
 - `__HAL_RCC_ADC1_CLK_ENABLE();`
 - `HAL_RCC_GetOscConfig(&RCC_OscInitStructure);`
 - `RCC_OscInitStructure.OscillatorType = (... | RCC_OSCILLATORTYPE_HSI);`
 - `RCC_OscInitStructure.HSIState = RCC_HSI_ON;`
 - `RCC_OscInitStructure.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;`
 - `RCC_OscInitStructure.PLL.PLLState = RCC_PLL_NONE;`
 - `RCC_OscInitStructure.PLL.PLLSource = ...`
 - `RCC_OscInitStructure.PLL...`
 - `HAL_RCC_OscConfig(&RCC_OscInitStructure);`
 - ADC clock prescaler is configured at ADC level with parameter "ClockPrescaler" using function HAL_ADC_Init().
2. ADC pins configuration
 - Enable the clock for the ADC GPIOs using macro `__HAL_RCC_GPIOx_CLK_ENABLE()`
 - Configure these ADC pins in analog mode using function HAL_GPIO_Init()
3. Optionally, in case of usage of ADC with interruptions:
 - Configure the NVIC for ADC using function HAL_NVIC_EnableIRQ(ADCx_IRQn)

- Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding ADC interruption vector ADCx_IRQHandler().
- 4. Optionally, in case of usage of DMA:
 - Configure the DMA (DMA channel, mode normal or circular, ...) using function HAL_DMA_Init().
 - Configure the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)
 - Insert the ADC interruption handler function HAL_ADC_IRQHandler() into the function of corresponding DMA interruption vector DMAx_Channelx_IRQHandler().

Configuration of ADC, groups regular/injected, channels parameters

1. Configure the ADC parameters (resolution, data alignment, ...) and regular group parameters (conversion trigger, sequencer, ...) using function HAL_ADC_Init().
2. Configure the channels for regular group parameters (channel number, channel rank into sequencer, ..., into regular group) using function HAL_ADC_ConfigChannel().
3. Optionally, configure the injected group parameters (conversion trigger, sequencer, ..., of injected group) and the channels for injected group parameters (channel number, channel rank into sequencer, ..., into injected group) using function HAL_ADCEx_InjectedConfigChannel().
4. Optionally, configure the analog watchdog parameters (channels monitored, thresholds, ...) using function HAL_ADC_AnalogWDGConfig().
5. Optionally, for devices with several ADC instances: configure the multimode parameters using function HAL_ADCEx_MultiModeConfigChannel().

Execution of ADC conversions

1. Optionally, perform an automatic ADC calibration to improve the conversion accuracy using function HAL_ADCEx_Calibration_Start().
2. ADC driver can be used among three modes: polling, interruption, transfer by DMA.
 - ADC conversion by polling:
 - Activate the ADC peripheral and start conversions using function HAL_ADC_Start()
 - Wait for ADC conversion completion using function HAL_ADC_PollForConversion() (or for injected group: HAL_ADCEx_InjectedPollForConversion())
 - Retrieve conversion results using function HAL_ADC_GetValue() (or for injected group: HAL_ADCEx_InjectedGetValue())
 - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop()
 - ADC conversion by interruption:
 - Activate the ADC peripheral and start conversions using function HAL_ADC_Start_IT()
 - Wait for ADC conversion completion by call of function HAL_ADC_ConvCpltCallback() (this function must be implemented in user program) (or for injected group: HAL_ADCEx_InjectedConvCpltCallback())
 - Retrieve conversion results using function HAL_ADC_GetValue() (or for injected group: HAL_ADCEx_InjectedGetValue())
 - Stop conversion and disable the ADC peripheral using function HAL_ADC_Stop_IT()
 - ADC conversion with transfer by DMA:

- Activate the ADC peripheral and start conversions using function `HAL_ADC_Start_DMA()`
- Wait for ADC conversion completion by call of function `HAL_ADC_ConvCpltCallback()` or `HAL_ADC_ConvHalfCpltCallback()` (these functions must be implemented in user program)
- Conversion results are automatically transferred by DMA into destination variable address.
- Stop conversion and disable the ADC peripheral using function `HAL_ADC_Stop_DMA()`
- For devices with several ADCs: ADC multimode conversion with transfer by DMA:
 - Activate the ADC peripheral (slave) and start conversions using function `HAL_ADC_Start()`
 - Activate the ADC peripheral (master) and start conversions using function `HAL_ADCEX_MultiModeStart_DMA()`
 - Wait for ADC conversion completion by call of function `HAL_ADC_ConvCpltCallback()` or `HAL_ADC_ConvHalfCpltCallback()` (these functions must be implemented in user program)
 - Conversion results are automatically transferred by DMA into destination variable address.
 - Stop conversion and disable the ADC peripheral (master) using function `HAL_ADCEX_MultiModeStop_DMA()`
 - Stop conversion and disable the ADC peripheral (slave) using function `HAL_ADC_Stop_IT()`



Callback functions must be implemented in user program:

- `HAL_ADC_ErrorCallback()`
- `HAL_ADC_LevelOutOfWindowCallback()` (callback of analog watchdog)
- `HAL_ADC_ConvCpltCallback()`
- `HAL_ADC_ConvHalfCpltCallback`
- `HAL_ADCEX_InjectedConvCpltCallback()`

Deinitialization of ADC

1. Disable the ADC interface
 - ADC clock can be hard reset and disabled at RCC top level.
 - Hard reset of ADC peripherals using macro `__ADCx_FORCE_RESET()`, `__ADCx_RELEASE_RESET()`.
 - ADC clock disable using the equivalent macro/functions as configuration step.
 - Example: Into `HAL_ADC_MspDeInit()` (recommended code location) or with other device clock parameters configuration:
 - `HAL_RCC_GetOscConfig(&RCC_OscInitStructure);`
 - `RCC_OscInitStructure.OscillatorType = RCC_OSCILLATORTYPE_HSI;`
 - `RCC_OscInitStructure.HSIState = RCC_HSI_OFF;` (if not used for system clock)
 - `HAL_RCC_OscConfig(&RCC_OscInitStructure);`
2. ADC pins configuration
 - Disable the clock for the ADC GPIOs using macro `__HAL_RCC_GPIOx_CLK_DISABLE()`
3. Optionally, in case of usage of ADC with interruptions:
 - Disable the NVIC for ADC using function `HAL_NVIC_EnableIRQ(ADCx_IRQn)`

4. Optionally, in case of usage of DMA:
 - Deinitialize the DMA using function HAL_DMA_Init().
 - Disable the NVIC for DMA using function HAL_NVIC_EnableIRQ(DMAx_Channelx_IRQn)

4.3.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the ADC.
- De-initialize the ADC.

This section contains the following APIs:

- [*HAL_ADC_Init\(\)*](#)
- [*HAL_ADC_DeInit\(\)*](#)
- [*HAL_ADC_MspInit\(\)*](#)
- [*HAL_ADC_MspDeInit\(\)*](#)

4.3.4 IO operation functions

This section provides functions allowing to:

- Start conversion of regular group.
- Stop conversion of regular group.
- Poll for conversion complete on regular group.
- Poll for conversion event.
- Get result of regular channel conversion.
- Start conversion of regular group and enable interruptions.
- Stop conversion of regular group and disable interruptions.
- Handle ADC interrupt request
- Start conversion of regular group and enable DMA transfer.
- Stop conversion of regular group and disable ADC DMA transfer.

This section contains the following APIs:

- [*HAL_ADC_Start\(\)*](#)
- [*HAL_ADC_Stop\(\)*](#)
- [*HAL_ADC_PollForConversion\(\)*](#)
- [*HAL_ADC_PollForEvent\(\)*](#)
- [*HAL_ADC_Start_IT\(\)*](#)
- [*HAL_ADC_Stop_IT\(\)*](#)
- [*HAL_ADC_Start_DMA\(\)*](#)
- [*HAL_ADC_Stop_DMA\(\)*](#)
- [*HAL_ADC_GetValue\(\)*](#)
- [*HAL_ADC_IRQHandler\(\)*](#)
- [*HAL_ADC_ConvCpltCallback\(\)*](#)
- [*HAL_ADC_ConvHalfCpltCallback\(\)*](#)
- [*HAL_ADC_LevelOutOfWindowCallback\(\)*](#)
- [*HAL_ADC_ErrorCallback\(\)*](#)

4.3.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on regular group
- Configure the analog watchdog

This section contains the following APIs:

- [HAL_ADC_ConfigChannel\(\)](#)
- [HAL_ADC_AnalogWDGConfig\(\)](#)

4.3.6 Peripheral State and Errors functions

This subsection provides functions to get in run-time the status of the peripheral.

- Check the ADC state
- Check the ADC error code

This section contains the following APIs:

- [HAL_ADC_GetState\(\)](#)
- [HAL_ADC_GetError\(\)](#)

4.3.7 HAL_ADC_Init

Function Name	HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef * hadc)
Function Description	Initializes the ADC peripheral and regular group according to parameters specified in structure "ADC_InitTypeDef".
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • As prerequisite, ADC clock must be configured at RCC top level (clock source APB2). See commented example code below that can be copied and uncommented into HAL_ADC_MspInit(). • Possibility to update parameters on the fly: This function initializes the ADC MSP (HAL_ADC_MspInit()) only when coming from ADC state reset. Following calls to this function can be used to reconfigure some parameters of ADC_InitTypeDef structure on the fly, without modifying MSP configuration. If ADC MSP has to be modified again, HAL_ADC_DeInit() must be called before HAL_ADC_Init(). The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_InitTypeDef". • This function configures the ADC within 2 scopes: scope of entire ADC and scope of regular group. For parameters details, see comments of structure "ADC_InitTypeDef".

4.3.8 HAL_ADC_DeInit

Function Name	HAL_StatusTypeDef HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)
Function Description	Deinitialize the ADC peripheral registers to its default reset values.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • To not impact other ADCs, reset of common ADC registers have been left commented below. If needed, the example code can be copied and uncommented into function

HAL_ADC_MspDeInit().

4.3.9 HAL_ADC_MspInit

Function Name	void HAL_ADC_MspInit (ADC_HandleTypeDef * hadc)
Function Description	Initializes the ADC MSP.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • None

4.3.10 HAL_ADC_MspDeInit

Function Name	void HAL_ADC_MspDeInit (ADC_HandleTypeDef * hadc)
Function Description	DeInitializes the ADC MSP.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • None

4.3.11 HAL_ADC_Start

Function Name	HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)
Function Description	Enables ADC, starts conversion of regular group.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • HAL status

4.3.12 HAL_ADC_Stop

Function Name	HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)
Function Description	Stop ADC conversion of regular group (and injected channels in case of auto_injection mode), disable ADC peripheral.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • HAL status.
Notes	<ul style="list-style-type: none"> • : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.

4.3.13 HAL_ADC_PollForConversion

Function Name	HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)
Function Description	Wait for regular group conversion to be completed.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle • Timeout: Timeout value in millisecond.
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- ADC conversion flags EOS (end of sequence) and EOC (end of conversion) are cleared by this function, with an exception: if low power feature "LowPowerAutoWait" is enabled, flags are not cleared to not interfere with this feature until data register is read using function HAL_ADC_GetValue().
- This function cannot be used in a particular setup: ADC configured in DMA mode and polling for end of each conversion (ADC init parameter "EOCSelection" set to ADC_EOC_SINGLE_CONV). In this case, DMA resets the flag EOC and polling cannot be performed on each conversion. Nevertheless, polling can still be performed on the complete sequence (ADC init parameter "EOCSelection" set to ADC_EOC_SEQ_CONV).

4.3.14 HAL_ADC_PollForEvent

Function Name

HAL_StatusTypeDef HAL_ADC_PollForEvent
(ADC_HandleTypeDef * hadc, uint32_t EventType, uint32_t Timeout)

Function Description

Poll for conversion event.

Parameters

- **hadc:** ADC handle
- **EventType:** the ADC event type. This parameter can be one of the following values: ADC_AWD_EVENT: ADC Analog watchdog event. ADC_OVR_EVENT: ADC Overrun event
- **Timeout:** Timeout value in millisecond.

Return values

- HAL status

4.3.15 HAL_ADC_Start_IT

Function Name

HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)

Function Description

Enables ADC, starts conversion of regular group with interruption.

4.3.16 HAL_ADC_Stop_IT

Function Name

HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)

Function Description

Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable interruption of end-of-conversion, disable ADC peripheral.

Parameters

- **hadc:** ADC handle

Return values

- None

4.3.17 HAL_ADC_Start_DMA

Function Name

HAL_StatusTypeDef HAL_ADC_Start_DMA
(ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t Length)

Function Description

Enables ADC, starts conversion of regular group and transfers result through DMA.

4.3.18 HAL_ADC_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)
Function Description	Stop ADC conversion of regular group (and injected group in case of auto_injection mode), disable ADC DMA transfer, disable ADC peripheral.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• HAL status.
Notes	<ul style="list-style-type: none">• : ADC peripheral disable is forcing stop of potential conversion on injected group. If injected group is under use, it should be preliminarily stopped using HAL_ADCEx_InjectedStop function.

4.3.19 HAL_ADC_GetValue

Function Name	uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)
Function Description	Get ADC regular group conversion result.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• Converted value
Notes	<ul style="list-style-type: none">• Reading DR register automatically clears EOC (end of conversion of regular group) flag.

4.3.20 HAL_ADC_IRQHandler

Function Name	void HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)
Function Description	Handles ADC interrupt request.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• None

4.3.21 HAL_ADC_ConvCpltCallback

Function Name	void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)
Function Description	Conversion complete callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• None

4.3.22 HAL_ADC_ConvHalfCpltCallback

Function Name	void HAL_ADC_ConvHalfCpltCallback (ADC_HandleTypeDef * hadc)
Function Description	Conversion DMA half-transfer callback in non blocking mode.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• None

4.3.23 HAL_ADC_LevelOutOfWindowCallback

Function Name	void HAL_ADC_LevelOutOfWindowCallback (ADC_HandleTypeDef * hadc)
Function Description	Analog watchdog callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • None

4.3.24 HAL_ADC_ErrorCallback

Function Name	void HAL_ADC_ErrorCallback (ADC_HandleTypeDef * hadc)
Function Description	ADC error callback in non blocking mode (ADC conversion with interruption or transfer by DMA)
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • None

4.3.25 HAL_ADC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)
Function Description	Configures the the selected channel to be linked to the regular group.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle • sConfig: Structure of ADC channel for regular group.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • In case of usage of internal measurement channels: Vbat/VrefInt/TempSensor. These internal paths can be disabled using function HAL_ADC_DeInit(). • Possibility to update parameters on the fly: This function initializes channel into regular group, following calls to this function can be used to reconfigure some parameters of structure "ADC_ChannelConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state. For parameters constraints, see comments of structure "ADC_ChannelConfTypeDef".

4.3.26 HAL_ADC_AnalogWDGConfig

Function Name	HAL_StatusTypeDef HAL_ADC_AnalogWDGConfig (ADC_HandleTypeDef * hadc, ADC_AnalogWDGConfTypeDef * AnalogWDGConfig)
Function Description	Configures the analog watchdog.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle • AnalogWDGConfig: Structure of ADC analog watchdog configuration
Return values	<ul style="list-style-type: none"> • HAL status

4.3.27 HAL_ADC_GetState

Function Name	uint32_t HAL_ADC_GetState (ADC_HandleTypeDef * hadc)
Function Description	return the ADC state
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• HAL state

4.3.28 HAL_ADC_GetError

Function Name	uint32_t HAL_ADC_GetError (ADC_HandleTypeDef * hadc)
Function Description	Return the ADC error code.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• ADC Error Code

4.4 ADC Firmware driver defines

4.4.1 ADC

ADC analog watchdog mode

ADC_ANALOGWATCHDOG_NONE
ADC_ANALOGWATCHDOG_SINGLE_REG
ADC_ANALOGWATCHDOG_SINGLE_INJEC
ADC_ANALOGWATCHDOG_SINGLE_REGINJEC
ADC_ANALOGWATCHDOG_ALL_REG
ADC_ANALOGWATCHDOG_ALL_INJEC
ADC_ANALOGWATCHDOG_ALL_REGINJEC

ADC channels

ADC_CHANNEL_0
ADC_CHANNEL_1
ADC_CHANNEL_2
ADC_CHANNEL_3
ADC_CHANNEL_4
ADC_CHANNEL_5
ADC_CHANNEL_6
ADC_CHANNEL_7
ADC_CHANNEL_8
ADC_CHANNEL_9
ADC_CHANNEL_10
ADC_CHANNEL_11
ADC_CHANNEL_12

ADC_CHANNEL_13
ADC_CHANNEL_14
ADC_CHANNEL_15
ADC_CHANNEL_16
ADC_CHANNEL_17
ADC_CHANNEL_18
ADC_CHANNEL_19
ADC_CHANNEL_20
ADC_CHANNEL_21
ADC_CHANNEL_22
ADC_CHANNEL_23
ADC_CHANNEL_24
ADC_CHANNEL_25
ADC_CHANNEL_26
ADC_CHANNEL_27
ADC_CHANNEL_28
ADC_CHANNEL_29
ADC_CHANNEL_30
ADC_CHANNEL_31
ADC_CHANNEL_TEMPSENSOR
ADC_CHANNEL_VREFINT
ADC_CHANNEL_VCOMP
ADC_CHANNEL_VOPAMP1
ADC_CHANNEL_VOPAMP2
ADC_CHANNEL_VOPAMP3

ADC ChannelsBank

ADC_CHANNELS_BANK_A
ADC_CHANNELS_BANK_B
IS_ADC_CHANNELSBANK
IS_ADC_CHANNELSBANK

ADC ClockPrescaler

ADC_CLOCK_ASYNC_DIV1	ADC asynchronous clock derived from ADC dedicated HSI without prescaler
ADC_CLOCK_ASYNC_DIV2	ADC asynchronous clock derived from ADC dedicated HSI divided by a prescaler of 2
ADC_CLOCK_ASYNC_DIV4	ADC asynchronous clock derived from ADC dedicated HSI divided by a prescaler of 4

ADC conversion group

ADC_REGULAR_GROUP

ADC_INJECTED_GROUP

ADC_REGULAR_INJECTED_GROUP

ADC Data_align

ADC_DATAALIGN_RIGHT

ADC_DATAALIGN_LEFT

ADC EOCSelection

ADC_EOC_SEQ_CONV

ADC_EOC_SINGLE_CONV

ADC Error Code

HAL_ADC_ERROR_NONE No error

HAL_ADC_ERROR_INTERNAL ADC IP internal error: if problem of clocking, enable/disable, erroneous state

HAL_ADC_ERROR_OVR Overrun error

HAL_ADC_ERROR_DMA DMA transfer error

ADC Event type

ADC_AWD_EVENT ADC Analog watchdog event

ADC_OVR_EVENT ADC overrun event

ADC Exported Macros

__HAL_ADC_ENABLE

Description:

- Enable the ADC peripheral.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_DISABLE

Description:

- Disable the ADC peripheral.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

__HAL_ADC_ENABLE_IT

Description:

- Enable the ADC end of conversion interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be any combination of the

following values:

- ADC_IT_EOC: ADC End of Regular Conversion interrupt source
- ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
- ADC_IT_AWD: ADC Analog watchdog interrupt source
- ADC_IT_OVR: ADC overrun interrupt source

Return value:

- None

Description:

- Disable the ADC end of conversion interrupt.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC Interrupt This parameter can be any combination of the following values:
 - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
 - ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
 - ADC_IT_AWD: ADC Analog watchdog interrupt source
 - ADC_IT_OVR: ADC overrun interrupt source

Return value:

- None

Description:

- Checks if the specified ADC interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: ADC handle
- __INTERRUPT__: ADC interrupt source to check This parameter can be any combination of the following values:
 - ADC_IT_EOC: ADC End of Regular Conversion interrupt source
 - ADC_IT_JEOC: ADC End of Injected Conversion interrupt source
 - ADC_IT_AWD: ADC Analog watchdog interrupt source
 - ADC_IT_OVR: ADC overrun interrupt source

Return value:

__HAL_ADC_DISABLE_IT

__HAL_ADC_GET_IT_SOURCE

`__HAL_ADC_GET_FLAG`

- State: of interruption (SET or RESET)

Description:

- Get the selected ADC's flag status.

Parameters:

- `__HANDLE__`: ADC handle
- `__FLAG__`: ADC flag This parameter can be any combination of the following values:
 - `ADC_FLAG_STRT`: ADC Regular group start flag
 - `ADC_FLAG_JSTRT`: ADC Injected group start flag
 - `ADC_FLAG_EOC`: ADC End of Regular conversion flag
 - `ADC_FLAG_JEOC`: ADC End of Injected conversion flag
 - `ADC_FLAG_AWD`: ADC Analog watchdog flag
 - `ADC_FLAG_OVR`: ADC overrun flag
 - `ADC_FLAG_ADONS`: ADC ready status flag
 - `ADC_FLAG_RCNR`: ADC Regular group ready status flag
 - `ADC_FLAG_JCNR`: ADC Injected group ready status flag

Return value:

- None

`__HAL_ADC_CLEAR_FLAG`**Description:**

- Clear the ADC's pending flags.

Parameters:

- `__HANDLE__`: ADC handle
- `__FLAG__`: ADC flag
 - `ADC_FLAG_STRT`: ADC Regular group start flag
 - `ADC_FLAG_JSTRT`: ADC Injected group start flag
 - `ADC_FLAG_EOC`: ADC End of Regular conversion flag
 - `ADC_FLAG_JEOC`: ADC End of Injected conversion flag
 - `ADC_FLAG_AWD`: ADC Analog watchdog flag
 - `ADC_FLAG_OVR`: ADC overrun flag
 - `ADC_FLAG_ADONS`: ADC ready status flag
 - `ADC_FLAG_RCNR`: ADC Regular group ready status flag
 - `ADC_FLAG_JCNR`: ADC Injected group ready status flag

__HAL_ADC_RESET_HANDLE_STATE

Return value:

- None

Description:

- Reset ADC handle state.

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

ADC Exported Types

HAL_ADC_STATE_RESET	ADC not yet initialized or disabled
HAL_ADC_STATE_READY	ADC peripheral ready for use
HAL_ADC_STATE_BUSY_INTERNAL	ADC is busy to internal process (initialization, calibration)
HAL_ADC_STATE_TIMEOUT	TimeOut occurrence
HAL_ADC_STATE_ERROR_INTERNAL	Internal error occurrence
HAL_ADC_STATE_ERROR_CONFIG	Configuration error occurrence
HAL_ADC_STATE_ERROR_DMA	DMA error occurrence
HAL_ADC_STATE_REG_BUSY	A conversion on group regular is ongoing or can occur (either by continuous mode, external trigger, low power auto power-on, multimode ADC master control)
HAL_ADC_STATE_REG_EOC	Conversion data available on group regular
HAL_ADC_STATE_REG_OVR	Overrun occurrence
HAL_ADC_STATE_REG_EOSMP	Not available on STM32L1 device: End Of Sampling flag raised
HAL_ADC_STATE_INJ_BUSY	A conversion on group injected is ongoing or can occur (either by auto-injection mode, external trigger, low power auto power-on, multimode ADC master control)
HAL_ADC_STATE_INJ_EOC	Conversion data available on group injected
HAL_ADC_STATE_INJ_JQOVF	Not available on STM32L1 device: Injected queue overflow occurrence
HAL_ADC_STATE_AWD1	Out-of-window occurrence of analog watchdog 1
HAL_ADC_STATE_AWD2	Not available on STM32L1 device: Out-of-window occurrence of analog watchdog 2
HAL_ADC_STATE_AWD3	Not available on STM32L1 device: Out-of-window occurrence of analog watchdog 3
HAL_ADC_STATE_MULTIMODE_SLAVE	Not available on STM32L1 device: ADC in multimode slave state, controlled by another ADC master (

ADC external trigger enable for regular group

ADC_EXTERNALTRIGCONVEDGE_NONE
ADC_EXTERNALTRIGCONVEDGE_RISING
ADC_EXTERNALTRIGCONVEDGE_FALLING
ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING

ADC External trigger source Regular

ADC_EXTERNALTRIGCONV_T2_CC3
ADC_EXTERNALTRIGCONV_T2_CC2
ADC_EXTERNALTRIGCONV_T2_TRGO
ADC_EXTERNALTRIGCONV_T3_CC1
ADC_EXTERNALTRIGCONV_T3_CC3
ADC_EXTERNALTRIGCONV_T3_TRGO
ADC_EXTERNALTRIGCONV_T4_CC4
ADC_EXTERNALTRIGCONV_T4_TRGO
ADC_EXTERNALTRIGCONV_T6_TRGO
ADC_EXTERNALTRIGCONV_T9_CC2
ADC_EXTERNALTRIGCONV_T9_TRGO
ADC_EXTERNALTRIGCONV_EXT_IT11
ADC_SOFTWARE_START

ADC flags definition

ADC_FLAG_AWD	ADC Analog watchdog flag
ADC_FLAG_EOC	ADC End of Regular conversion flag
ADC_FLAG_JEOC	ADC End of Injected conversion flag
ADC_FLAG_JSTRT	ADC Injected group start flag
ADC_FLAG_STRT	ADC Regular group start flag
ADC_FLAG_OVR	ADC overrun flag
ADC_FLAG_ADONS	ADC ready status flag
ADC_FLAG_RCNR	ADC Regular group ready status flag
ADC_FLAG_JCNR	ADC Injected group ready status flag

ADC Internal HAL driver Ext trig src Regular

ADC_EXTERNALTRIG_T9_CC2
ADC_EXTERNALTRIG_T9_TRGO
ADC_EXTERNALTRIG_T2_CC3
ADC_EXTERNALTRIG_T2_CC2
ADC_EXTERNALTRIG_T3_TRGO
ADC_EXTERNALTRIG_T4_CC4

ADC EXTERNALTRIG T2 TRGO

ADC_EXTERNALTRIG_T3_CC1

ADC EXTERNALTRIG T3 CC3

ADC EXTERNALTRIG T4 TRGO

ADC_EXTERNALTRIG_T6_TRGO

ADC_EXTERNALTRIG_EXT_IT11

ADC interrupts definition

ADC_IT_EOC	ADC End of Regular Conversion interrupt source
------------	--

ADC IT JEOC ADC End of Injected Conversion interrupt source

ADC IT AWD ADC Analog watchdog interrupt source

ADC IT OVR ADC overrun interrupt source

ADC LowPowerAutoPowerOff

ADC_AUTOPOWEROFF_DISABLE

ADC_AUTOPOWEROFF_IDLE_PHASE	ADC power off when ADC is not converting (idle phase)
-----------------------------	---

ADC_AUTOPOWEROFF_DELAY_PHASE	ADC power off when a delay is inserted between conversions (see parameter ADC_LowPowerAutoWait)
------------------------------	---

ADC_AUTOPOWEROFF_IDLE_DELAY_PHASES	ADC power off when ADC is not converting (idle phase) and when a delay is inserted between conversions
------------------------------------	--

ADC LowPowerAutoWait

ADC_AUTOWAIT_DISABLE < Note : For compatibility with other STM32 devices with ADC autowait

ADC_AUTOWAIT_UNTIL_DATA_READ	Insert a delay between ADC conversions: infinite delay, until the result of previous conversion is read
------------------------------	--

ADC_AUTOWAIT_7_APBCLKCYCLES	Insert a delay between ADC conversions: 7 APB clock cycles
-----------------------------	---

ADC_AUTOWAIT_15_APBCLKCYCLES	Insert a delay between ADC conversions: 15 APB clock cycles
------------------------------	--

ADC_AUTOWAIT_31_APBCLKCYCLES	Insert a delay between ADC conversions: 31 APB clock cycles
------------------------------	--

ADC_AUTOWAIT_63_APBCLKCYCLES	Insert a delay between ADC conversions: 63 APB clock cycles
------------------------------	--

ADC_AUTOWAIT_127_APBCLKCYCLES	Insert a delay between ADC conversions: 127 APB clock cycles
-------------------------------	---

ADC_AUTOWAIT_255_APBCLKCYCLES	Insert a delay between ADC conversions: 255 APB clock cycles
-------------------------------	---

ADC Private Constants

ADC_ENABLE_TIMEOUT

ADC_DISABLE_TIMEOUT

ADC_STAB_DELAY_US

ADC_TEMPSENSOR_DELAY_US

ADC_FLAG_POSTCONV_ALL

ADC Private Macros

ADC_IS_ENABLE

Description:

- Verification of ADC state: enabled or disabled.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

- SET: (ADC enabled) or RESET (ADC disabled)

ADC_IS_SOFTWARE_START_REGULAR

Description:

- Test if conversion trigger of regular group is software start or external trigger.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

- SET: (software start) or RESET (external trigger)

ADC_IS_SOFTWARE_START_INJECTED

Description:

- Test if conversion trigger of injected group is software start or external trigger.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

- SET: (software start) or RESET (external trigger)

ADC_STATE_CLR_SET

Description:

- Simultaneously clears and sets specific bits of the handle State.

Return value:

- None

Notes:

- `ADC_STATE_CLR_SET()` macro is merely aliased to generic macro

MODIFY_REG(), the first parameter is the ADC handle State, the second parameter is the bit field to clear, the third and last parameter is the bit field to set.

ADC_CLEAR_ERRORCODE

Description:

- Clear ADC error code (set it to error code: "no error")

Parameters:

- __HANDLE__: ADC handle

Return value:

- None

ADC_SQR1_L_SHIFT

Description:

- Set ADC number of ranks into regular channel sequence length.

Parameters:

- _NbrOfConversion_: Regular channel sequence length

Return value:

- None

ADC_SMPR2

Description:

- Set the ADC's sample time for channel numbers between 10 and 18.

Parameters:

- _SAMPLETIME_: Sample time parameter.
- _CHANNELNB_: Channel number.

Return value:

- None

ADC_SMPR3

Description:

- Set the ADC's sample time for channel numbers between 0 and 9.

Parameters:

- _SAMPLETIME_: Sample time parameter.
- _CHANNELNB_: Channel number.

Return value:

- None

ADC_SQR5_RK

Description:

- Set the selected regular channel rank for rank between 1 and 6.

ADC_SQR4_RK

Parameters:

- `_CHANNELNB_`: Channel number.
- `_RANKNB_`: Rank number.

Return value:

- None

Description:

- Set the selected regular channel rank for rank between 7 and 12.

Parameters:

- `_CHANNELNB_`: Channel number.
- `_RANKNB_`: Rank number.

Return value:

- None

Description:

- Set the selected regular channel rank for rank between 13 and 18.

Parameters:

- `_CHANNELNB_`: Channel number.
- `_RANKNB_`: Rank number.

Return value:

- None

Description:

- Set the selected regular channel rank for rank between 19 and 24.

Parameters:

- `_CHANNELNB_`: Channel number.
- `_RANKNB_`: Rank number.

Return value:

- None

Description:

- Set the selected regular channel rank for rank between 25 and 28.

Parameters:

- `_CHANNELNB_`: Channel number.
- `_RANKNB_`: Rank number.

Return value:

- None

Description:

- Set the injected sequence length.

ADC_SQR3_RK

ADC_SQR2_RK

ADC_SQR1_RK

ADC_JSQR_JL_SHIFT

ADC_JSQR_RK_JL

Parameters:

- `_JSQR_JL_`: Sequence length.

Return value:

- None

Description:

- Set the selected injected channel rank
Note: on STM32L1 devices, channel rank position in JSQR register is depending on total number of ranks selected into injected sequencer (ranks sequence starting from 4-JL)

Parameters:

- `_CHANNELNB_`: Channel number.
- `_RANKNB_`: Rank number.
- `_JSQR_JL_`: Sequence length.

Return value:

- None

ADC_CR2_DMACONTREQ

Description:

- Enable the ADC DMA continuous request.

Parameters:

- `_DMACONTREQ_MODE_`: DMA continuous request mode.

Return value:

- None

ADC_CR2_CONTINUOUS

Description:

- Enable ADC continuous conversion mode.

Parameters:

- `_CONTINUOUS_MODE_`: Continuous mode.

Return value:

- None

ADC_CR1_DISCONTINUOUS_NUM

Description:

- Configures the number of discontinuous conversions for the regular group channels.

Parameters:

- `_NBR_DISCONTINUOUS_CONV_`: Number of discontinuous conversions.

Return value:

ADC_CR1_SCAN_SET

- None

Description:

- Enable ADC scan mode to convert multiple ranks with sequencer.

Parameters:

- `_SCAN_MODE_`: Scan conversion mode.

Return value:

- None

IS_ADC_CLOCKPRESCALER

IS_ADC_RESOLUTION

IS_ADC_RESOLUTION_8_6_BITS

IS_ADC_DATA_ALIGN

IS_ADC_SCAN_MODE

IS_ADC_EXTTRIG_EDGE

IS_ADC_EXTTRIG

IS_ADC_EOC_SELECTION

IS_ADC_AUTOWAIT

IS_ADC_AUTOPOWEROFF

IS_ADC_CHANNEL

IS_ADC_SAMPLE_TIME

IS_ADC_REGULAR_RANK

IS_ADC_ANALOG_WATCHDOG_MODE

IS_ADC_CONVERSION_GROUP

IS_ADC_EVENT_TYPE

ADC range verification

IS_ADC_RANGE

ADC regular discontinuous mode number verification

IS_ADC_REGULAR_DISCONT_NUMBER

ADC regular nb conv verification

IS_ADC_REGULAR_NB_CONV

ADC rank into regular group

ADC_REGULAR_RANK_1

ADC_REGULAR_RANK_2

ADC_REGULAR_RANK_3

ADC_REGULAR_RANK_4

ADC_REGULAR_RANK_5

ADC_REGULAR_RANK_6
ADC_REGULAR_RANK_7
ADC_REGULAR_RANK_8
ADC_REGULAR_RANK_9
ADC_REGULAR_RANK_10
ADC_REGULAR_RANK_11
ADC_REGULAR_RANK_12
ADC_REGULAR_RANK_13
ADC_REGULAR_RANK_14
ADC_REGULAR_RANK_15
ADC_REGULAR_RANK_16
ADC_REGULAR_RANK_17
ADC_REGULAR_RANK_18
ADC_REGULAR_RANK_19
ADC_REGULAR_RANK_20
ADC_REGULAR_RANK_21
ADC_REGULAR_RANK_22
ADC_REGULAR_RANK_23
ADC_REGULAR_RANK_24
ADC_REGULAR_RANK_25
ADC_REGULAR_RANK_26
ADC_REGULAR_RANK_27
ADC_REGULAR_RANK_28

ADC Resolution

ADC_RESOLUTION_12B ADC 12-bit resolution
ADC_RESOLUTION_10B ADC 10-bit resolution
ADC_RESOLUTION_8B ADC 8-bit resolution
ADC_RESOLUTION_6B ADC 6-bit resolution

ADC sampling times

ADC_SAMPLETIME_4CYCLES Sampling time 4 ADC clock cycles
ADC_SAMPLETIME_9CYCLES Sampling time 9 ADC clock cycles
ADC_SAMPLETIME_16CYCLES Sampling time 16 ADC clock cycles
ADC_SAMPLETIME_24CYCLES Sampling time 24 ADC clock cycles
ADC_SAMPLETIME_48CYCLES Sampling time 48 ADC clock cycles
ADC_SAMPLETIME_96CYCLES Sampling time 96 ADC clock cycles
ADC_SAMPLETIME_192CYCLES Sampling time 192 ADC clock cycles

ADC_SAMPLETIME_384CYCLES Sampling time 384 ADC clock cycles

ADC sampling times all channels

ADC_SAMPLETIME_ALLCHANNELS_SMPR3BIT2

ADC_SAMPLETIME_ALLCHANNELS_SMPR2BIT2

ADC_SAMPLETIME_ALLCHANNELS_SMPR1BIT2

ADC_SAMPLETIME_ALLCHANNELS_SMPR0BIT2

ADC_SAMPLETIME_ALLCHANNELS_SMPR3BIT1

ADC_SAMPLETIME_ALLCHANNELS_SMPR2BIT1

ADC_SAMPLETIME_ALLCHANNELS_SMPR1BIT1

ADC_SAMPLETIME_ALLCHANNELS_SMPR0BIT1

ADC_SAMPLETIME_ALLCHANNELS_SMPR3BIT0

ADC_SAMPLETIME_ALLCHANNELS_SMPR2BIT0

ADC_SAMPLETIME_ALLCHANNELS_SMPR1BIT0

ADC_SAMPLETIME_ALLCHANNELS_SMPR0BIT0

ADC Scan mode

ADC_SCAN_DISABLE

ADC_SCAN_ENABLE

5 HAL ADC Extension Driver

5.1 HAL ADC Extension Driver

5.2 ADCEX Firmware driver registers structures

5.2.1 ADC_InjectionConfTypeDef

Data Fields

- *uint32_t InjectedChannel*
- *uint32_t InjectedRank*
- *uint32_t InjectedSamplingTime*
- *uint32_t InjectedOffset*
- *uint32_t InjectedNbrOfConversion*
- *uint32_t InjectedDiscontinuousConvMode*
- *uint32_t AutoInjectedConv*
- *uint32_t ExternalTrigInjecConv*
- *uint32_t ExternalTrigInjecConvEdge*

Field Documentation

- ***uint32_t ADC_InjectionConfTypeDef::InjectedChannel***
Selection of ADC channel to configure This parameter can be a value of [ADC_channels](#) Note: Depending on devices, some channels may not be available on package pins. Refer to device datasheet for channels availability.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedRank***
Rank in the injected group sequencer This parameter must be a value of [ADCEX_injected_rank](#) Note: In case of need to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by the new channel setting (or parameter number of conversions can be adjusted)
- ***uint32_t ADC_InjectionConfTypeDef::InjectedSamplingTime***
Sampling time value to be set for the selected channel. Unit: ADC clock cycles
Conversion time is the addition of sampling time and processing time (12 ADC clock cycles at ADC resolution 12 bits, 11 cycles at 10 bits, 9 cycles at 8 bits, 7 cycles at 6 bits). This parameter can be a value of [ADC_sampling_times](#) Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups. If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting. Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor), sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting) Refer to device datasheet for timings values, parameters TS_vrefint, TS_temp (values rough order: 4us min).
- ***uint32_t ADC_InjectionConfTypeDef::InjectedOffset***
Defines the offset to be subtracted from the raw converted data (for channels set on injected group only). Offset value must be a positive number. Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xFFF, 0x3FF, 0xFF or 0x3F respectively.
- ***uint32_t ADC_InjectionConfTypeDef::InjectedNbrOfConversion***
Specifies the number of ranks that will be converted within the injected group

sequencer. To use the injected group sequencer and convert several ranks, parameter 'ScanConvMode' must be enabled. This parameter must be a number between Min_Data = 1 and Max_Data = 4. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEX_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

- uint32_t ADC_InjectionConfTypeDef::InjectedDiscontinuousConvMode**
 Specifies whether the conversions sequence of injected group is performed in Complete-sequence/Discontinuous-sequence (main sequence subdivided in successive parts). Discontinuous mode is used only if sequencer is enabled (parameter 'ScanConvMode'). If sequencer is disabled, this parameter is discarded. Discontinuous mode can be enabled only if continuous mode is disabled. If continuous mode is enabled, this parameter setting is discarded. This parameter can be set to ENABLE or DISABLE. Note: For injected group, number of discontinuous ranks increment is fixed to one-by-one. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEX_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- uint32_t ADC_InjectionConfTypeDef::AutoInjectedConv**
 Enables or disables the selected ADC automatic injected group conversion after regular one. This parameter can be set to ENABLE or DISABLE. Note: To use Automatic injected conversion, discontinuous mode must be disabled ('DiscontinuousConvMode' and 'InjectedDiscontinuousConvMode' set to DISABLE). Note: To use Automatic injected conversion, injected group external triggers must be disabled ('ExternalTrigInjecConv' set to ADC_SOFTWARE_START). Note: In case of DMA used with regular group: if DMA configured in normal mode (single shot) JAUTO will be stopped upon DMA transfer complete. To maintain JAUTO always enabled, DMA must be configured in circular mode. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEX_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConv**
 Selects the external event used to trigger the conversion start of injected group. If set to ADC_INJECTED_SOFTWARE_START, external triggers are disabled. If set to external trigger source, triggering is on event rising edge. This parameter can be a value of [ADCEX_External_trigger_source_Injected](#). Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion). If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behaviour in case of another parameter update on the fly). Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEX_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.
- uint32_t ADC_InjectionConfTypeDef::ExternalTrigInjecConvEdge**
 Selects the external trigger edge of injected group. This parameter can be a value of [ADCEX_External_trigger_edge_Injected](#). If trigger is set to ADC_INJECTED_SOFTWARE_START, this parameter is discarded. Caution: this setting impacts the entire injected group. Therefore, call of **HAL_ADCEX_InjectedConfigChannel()** to configure a channel on injected group can impact the configuration of other channels previously set.

5.3 ADCEx Firmware driver API description

5.3.1 IO operation functions

This section provides functions allowing to:

- Start conversion of injected group.
- Stop conversion of injected group.
- Poll for conversion complete on injected group.
- Get result of injected channel conversion.
- Start conversion of injected group and enable interruptions.
- Stop conversion of injected group and disable interruptions.

This section contains the following APIs:

- [*HAL_ADCEx_InjectedStart\(\)*](#)
- [*HAL_ADCEx_InjectedStop\(\)*](#)
- [*HAL_ADCEx_InjectedPollForConversion\(\)*](#)
- [*HAL_ADCEx_InjectedStart_IT\(\)*](#)
- [*HAL_ADCEx_InjectedStop_IT\(\)*](#)
- [*HAL_ADCEx_InjectedGetValue\(\)*](#)
- [*HAL_ADCEx_InjectedConvCpltCallback\(\)*](#)

5.3.2 Peripheral Control functions

This section provides functions allowing to:

- Configure channels on injected group

This section contains the following APIs:

- [*HAL_ADCEx_InjectedConfigChannel\(\)*](#)

5.3.3 HAL_ADCEx_InjectedStart

Function Name **HAL_StatusTypeDef HAL_ADCEx_InjectedStart
(ADC_HandleTypeDef * hadc)**

Function Description Enables ADC, starts conversion of injected group.

Parameters • **hadc:** ADC handle

Return values • HAL status

5.3.4 HAL_ADCEx_InjectedStop

Function Name **HAL_StatusTypeDef HAL_ADCEx_InjectedStop
(ADC_HandleTypeDef * hadc)**

Function Description Stop conversion of injected channels.

Parameters • **hadc:** ADC handle

Return values • None

Notes

- If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.
- In case of auto-injection mode, HAL_ADC_Stop must be used.

5.3.5 HAL_ADCEx_InjectedPollForConversion

Function Name	HAL_StatusTypeDef HAL_ADCEx_InjectedPollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)
Function Description	Wait for injected group conversion to be completed.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle• Timeout: Timeout value in millisecond.
Return values	<ul style="list-style-type: none">• HAL status

5.3.6 HAL_ADCEx_InjectedStart_IT

Function Name	HAL_StatusTypeDef HAL_ADCEx_InjectedStart_IT (ADC_HandleTypeDef * hadc)
Function Description	Enables ADC, starts conversion of injected group with interruption.

5.3.7 HAL_ADCEx_InjectedStop_IT

Function Name	HAL_StatusTypeDef HAL_ADCEx_InjectedStop_IT (ADC_HandleTypeDef * hadc)
Function Description	Stop conversion of injected channels, disable interruption of end-of-conversion.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle
Return values	<ul style="list-style-type: none">• None
Notes	<ul style="list-style-type: none">• If ADC must be disabled and if conversion is on going on regular group, function HAL_ADC_Stop must be used to stop both injected and regular groups, and disable the ADC.

5.3.8 HAL_ADCEx_InjectedGetValue

Function Name	uint32_t HAL_ADCEx_InjectedGetValue (ADC_HandleTypeDef * hadc, uint32_t InjectedRank)
Function Description	Get ADC injected group conversion result.
Parameters	<ul style="list-style-type: none">• hadc: ADC handle• InjectedRank: the converted ADC injected rank. This parameter can be one of the following values: ADC_INJECTED_RANK_1: Injected Channel1 selectedADC_INJECTED_RANK_2: Injected Channel2 selectedADC_INJECTED_RANK_3: Injected Channel3 selectedADC_INJECTED_RANK_4: Injected Channel4 selected
Return values	<ul style="list-style-type: none">• None

5.3.9 HAL_ADCEx_InjectedConvCpltCallback

Function Name	void HAL_ADCEx_InjectedConvCpltCallback (ADC_HandleTypeDef * hadc)
Function Description	Injected conversion complete callback in non blocking mode.

Parameters	<ul style="list-style-type: none"> • hadc: ADC handle
Return values	<ul style="list-style-type: none"> • None

5.3.10 HAL_ADCEX_InjectedConfigChannel

Function Name	HAL_StatusTypeDef HAL_ADCEX_InjectedConfigChannel (ADC_HandleTypeDef * hadc, ADC_InjectionConfTypeDef * sConfigInjected)
Function Description	Configures the ADC injected group and the selected channel to be linked to the injected group.
Parameters	<ul style="list-style-type: none"> • hadc: ADC handle • sConfigInjected: Structure of ADC injected group and ADC channel for injected group.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • Possibility to update parameters on the fly: This function initializes injected group, following calls to this function can be used to reconfigure some parameters of structure "ADC_InjectionConfTypeDef" on the fly, without resetting the ADC. The setting of these parameters is conditioned to ADC state: this function must be called when ADC is not under conversion.

5.4 ADCEX Firmware driver defines

5.4.1 ADCEX

ADCEX Exported Macros

__HAL_ADC_CHANNELS_BANK	Description: <ul style="list-style-type: none"> • Selection of channels bank. Parameters: <ul style="list-style-type: none"> • __HANDLE__: ADC handle • __BANK__: Bank selection. This parameter can be a value of Return value: <ul style="list-style-type: none"> • None
__HAL_ADC_CHANNEL_SPEED_FAST	Limited to channels 3, 8, 13 and to devices category Cat.3, Cat.4, Cat.5.
__HAL_ADC_CHANNEL_SPEED_SLOW	
<i>ADCEX external trigger enable for injected group</i>	
ADC_EXTERNALTRIGINJECCONV_EDGE_NONE	
ADC_EXTERNALTRIGINJECCONV_EDGE_RISING	
ADC_EXTERNALTRIGINJECCONV_EDGE_FALLING	
ADC_EXTERNALTRIGINJECCONV_EDGE_RISINGFALLING	
<i>ADCEX External trigger source Injected</i>	

ADC_EXTERNALTRIGINJEC_CONV_T2_CC1
ADC_EXTERNALTRIGINJEC_CONV_T2_TRGO
ADC_EXTERNALTRIGINJEC_CONV_T3_CC4
ADC_EXTERNALTRIGINJEC_CONV_T4_TRGO
ADC_EXTERNALTRIGINJEC_CONV_T4_CC1
ADC_EXTERNALTRIGINJEC_CONV_T4_CC2
ADC_EXTERNALTRIGINJEC_CONV_T4_CC3
ADC_EXTERNALTRIGINJEC_CONV_T7_TRGO
ADC_EXTERNALTRIGINJEC_CONV_T9_CC1
ADC_EXTERNALTRIGINJEC_CONV_T9_TRGO
ADC_EXTERNALTRIGINJEC_CONV_T10_CC1
ADC_EXTERNALTRIGINJEC_CONV_EXT_IT15
ADC_INJECTED_SOFTWARE_START

ADCEX injected nb conv verification

IS_ADC_INJECTED_NB_CONV

ADCEX rank into injected group

ADC_INJECTED_RANK_1
ADC_INJECTED_RANK_2
ADC_INJECTED_RANK_3
ADC_INJECTED_RANK_4

ADCEX Internal HAL driver Ext trig src Injected

ADC_EXTERNALTRIGINJEC_T9_CC1
ADC_EXTERNALTRIGINJEC_T9_TRGO
ADC_EXTERNALTRIGINJEC_T2_TRGO
ADC_EXTERNALTRIGINJEC_T2_CC1
ADC_EXTERNALTRIGINJEC_T3_CC4
ADC_EXTERNALTRIGINJEC_T4_TRGO
ADC_EXTERNALTRIGINJEC_T4_CC1
ADC_EXTERNALTRIGINJEC_T4_CC2
ADC_EXTERNALTRIGINJEC_T4_CC3
ADC_EXTERNALTRIGINJEC_T10_CC1
ADC_EXTERNALTRIGINJEC_T7_TRGO
ADC_EXTERNALTRIGINJEC_EXT_IT15

ADCEX Private Constants

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_4CYCLE5
ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_9CYCLES

ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_16CYCLES
ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_24CYCLES
ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_48CYCLES
ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_96CYCLES
ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_192CYCLES
ADC_CONVERSIONCLOCKCYCLES_SAMPLETIME_384CYCLES
ADC_TEMPSENSOR_DELAY_US

ADCEx Private Macro

__ADC_SQR1_SQXX

Description:

- Set ADC ranks available in register SQR1.

Parameters:

- `_NbrOfConversion_`: Regular channel sequence length

Return value:

- None

ADC_SMPR0

Description:

- Set the ADC's sample time for channel numbers between 30 and 31.

Parameters:

- `_SAMPLETIME_`: Sample time parameter.
- `_CHANNELNB_`: Channel number.

Return value:

- None
- None

ADC_SMPR1

Description:

- Set the ADC's sample time for channel numbers between 20 and 29.

Parameters:

- `_SAMPLETIME_`: Sample time parameter.
- `_CHANNELNB_`: Channel number.

Return value:

- None

ADC_SMPR1_CHANNEL_MAX

Description:

- Defines the highest channel available in register SMPR1.

Parameters:

ADC_CR2_MASK_ADCINIT

- None

Return value:

- None

Description:

- Define mask of configuration bits of ADC and regular group in register CR2 (bits of ADC enable, conversion start and injected group are excluded of this mask).

Return value:

- None

ADC_CONVCYCLES_MAX_RANGE**Description:**

- Get the maximum ADC conversion cycles on all channels.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

- ADC: conversion cycles on all channels

ADC_GET_CLOCK_PRESCALER_DECIMAL**Description:**

- Get the ADC clock prescaler from ADC common control register and convert it to its decimal number setting (refer to reference manual)

Return value:

- None

ADC_SMPR0_CLEAR**Description:**

- Clear register SMPR0.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

- None

ADC_CR2_CLEAR**Description:**

- Clear register CR2.

Parameters:

- `__HANDLE__`: ADC handle

Return value:

- None

ADC_SMPR0_CHANNEL_SET**Description:**

- Set the sampling time of selected channel on register SMPR0 Register SMPR0 availability depends on device category.

Parameters:

- __HANDLE__: ADC handle
- _SAMPLETIME_: Sample time parameter.
- __CHANNEL__: Channel number.

Return value:

- None

IS_ADC_INJECTED_RANK
IS_ADC_EXTTRIGINJEC_EDGE
IS_ADC_EXTTRIGINJEC

6 HAL COMP Generic Driver

6.1 HAL COMP Generic Driver

6.2 COMP Firmware driver registers structures

6.2.1 COMP_InitTypeDef

Data Fields

- *uint32_t InvertingInput*
- *uint32_t NonInvertingInput*
- *uint32_t Output*
- *uint32_t Mode*
- *uint32_t WindowMode*
- *uint32_t TriggerMode*
- *uint32_t NonInvertingInputPull*

Field Documentation

- ***uint32_t COMP_InitTypeDef::InvertingInput***
Selects the inverting input of the comparator. This parameter can be a value of [COMP_InvertingInput](#) Note: Inverting input can be changed on the fly, while comparator is running. Note: This feature is available on COMP2 only. If COMP1 is selected, this parameter is discarded (On COMP1, inverting input is fixed to Vrefint).
- ***uint32_t COMP_InitTypeDef::NonInvertingInput***
Selects the non inverting input of the comparator. This parameter can be a value of [COMPEx_NonInvertingInput](#)
- ***uint32_t COMP_InitTypeDef::Output***
Selects the output redirection of the comparator. This parameter can be a value of [COMP_Output](#) Note: This feature is available on COMP2 only. If COMP1 is selected, this parameter is discarded.
- ***uint32_t COMP_InitTypeDef::Mode***
Selects the operating consumption mode of the comparator to adjust the speed/consumption. This parameter can be a value of [COMP_Mode](#) Note: This feature is available on COMP2 only. If COMP1 is selected, this parameter is discarded.
- ***uint32_t COMP_InitTypeDef::WindowMode***
Selects the window mode of the 2 comparators. If enabled, non-inverting inputs of the 2 comparators are connected together and are using inputs of COMP2 only (COMP1 non-inverting input is no more accessible, even from ADC channel VCOMP). This parameter can be a value of [COMP_WindowMode](#) Note: This feature must be enabled from COMP2 instance. If COMP1 is selected, this parameter is discarded.
- ***uint32_t COMP_InitTypeDef::TriggerMode***
Selects the trigger mode of the comparator when using interruption on EXTI line (interrupt mode). This parameter can be a value of [COMP_TriggerMode](#) Note: This feature is used with function "HAL_COMP_Start_IT()". In all other functions, this parameter is discarded.
- ***uint32_t COMP_InitTypeDef::NonInvertingInputPull***
Selects the internal pulling resistor connected on non inverting input. This parameter

can be a value of [COMP_NonInvertingInputPull](#) Note: To avoid extra power consumption, only one resistor should be enabled at a time. Note: This feature is available on COMP1 only. If COMP2 is selected, this parameter is discarded.

6.2.2 COMP_HandleTypeDef

Data Fields

- **COMP_TypeDef * Instance**
- **COMP_InitTypeDef Init**
- **HAL_LockTypeDef Lock**
- **__IO HAL_COMP_StateTypeDef State**

Field Documentation

- **COMP_TypeDef* COMP_HandleTypeDef::Instance**
Register base address
- **COMP_InitTypeDef COMP_HandleTypeDef::Init**
COMP required parameters
- **HAL_LockTypeDef COMP_HandleTypeDef::Lock**
Locking object
- **__IO HAL_COMP_StateTypeDef COMP_HandleTypeDef::State**
COMP communication state

6.3 COMP Firmware driver API description

6.3.1 COMP Peripheral features

The STM32L1xx device family integrates 2 analog comparators COMP1 and COMP2:

1. The non inverting input and inverting input can be set to GPIO pins. Refer to [Table 17: "COMP Inputs for the STM32L1xx devices"](#). HAL COMP driver configures the Routing Interface (RI) to connect the selected I/O pins to comparator input. Caution: Comparator COMP1 and ADC cannot be used at the same time as ADC since they share the ADC switch matrix: COMP1 non-inverting input is routed through ADC switch matrix. Except if ADC is intended to measure voltage on COMP1 non-inverting input: it can be performed on ADC channel VCOMP.
2. The COMP output is available using HAL_COMP_GetOutputLevel().
3. The COMP output can be redirected to embedded timers (TIM2, TIM3, TIM4, TIM10). COMP output cannot be redirected to any I/O pin. Refer to [Table 16: "Redirection of COMP outputs to embedded timers"](#).
4. The comparators COMP1 and COMP2 can be combined in window mode. In this mode, COMP2 non inverting input is used as common non-inverting input.
5. The 2 comparators have interrupt capability with wake-up from Sleep and Stop modes (through the EXTI controller):
 - COMP1 is internally connected to EXTI Line 21
 - COMP2 is internally connected to EXTI Line 22 From the corresponding IRQ handler, the right interrupt source can be retrieved with the macros `__HAL_COMP_COMP1_EXTI_GET_FLAG()` and `__HAL_COMP_COMP2_EXTI_GET_FLAG()`.

6. The comparators also offer the possibility to output the voltage reference (VrefInt), used on inverting inputs, on I/O pin through a buffer. To use it, refer to macro "`__HAL_SYSCFG_VREFINT_OUT_ENABLE()`".

Table 16: Redirection of COMP outputs to embedded timers

COMP1	COMP2
No redirection to timers	TIM2 IC4 TIM2 OCREF CLR TIM3 IC4 TIM3 OCREF CLR TIM4 IC4 TIM4 OCREF CLR TIM10 IC1

Table 17: COMP Inputs for the STM32L1xx devices

		COMP1	COMP2
Inverting inputs	1/4 VREFINT	-	OK
	1/2 VREFINT	-	OK
	3/4 VREFINT	-	OK
	VREFINT	OK	OK
	DAC Ch1 OUT (PA4)	-	OK
	DAC Ch2 OUT (PA5)	-	OK
	I/O: PB3	-	OK
Non-inverting inputs	I/O:	-	OK
	• PB4, 5, 6, 7 ⁽¹⁾	OK	-
	• PA0, 1, 2, 3, 4, 5, 6, 7 ⁽²⁾	OK	-
	• PB0, 1, 12, 13, 14, 15	OK	-
	• PC0 1, 2, 3, 4, 5	OK	-
	• PE7, 8, 9, 10	OK	-
	• PF6, 7, 8, 9, 10	OK	-
	• OPAMP1 output	OK	-
	• OPAMP2 output	OK	-
	• OPAMP3 output ⁽³⁾	OK	-

Notes:

⁽¹⁾PA6/7 are available on devices category Cat.3, Cat.4, Cat.5 only.

⁽²⁾PA0/1/2/3 are available on devices category Cat.3, Cat.4, Cat.5 only.

⁽³⁾Available on devices category Cat.4 only.

6.3.2 How to use this driver

This driver provides functions to configure and program the Comparators of all STM32L1xx devices. To use the comparator, perform the following steps:

1. Initialize the COMP low level resources by implementing the `HAL_COMP_MspInit()`.
 - Configure the comparator input I/O pin using `HAL_GPIO_Init()`: - For all inputs: I/O pin in analog mode (Schmitt trigger disabled) - Possible alternate configuration, for non-inverting inputs of comparator 2: I/O pin in floating mode

- (Schmitt trigger enabled). It is recommended to use analog configuration to avoid any overconsumption around VDD/2.
- Enable COMP Peripheral clock using macro `__HAL_RCC_COMP_CLK_ENABLE()`
 - If required enable the COMP interrupt (EXTI line Interrupt): enable the comparator interrupt vector using `HAL_NVIC_EnableIRQ(COMP_IRQn)` and `HAL_NVIC_SetPriority(COMP_IRQn, xxx, xxx)` functions.
2. Configure the comparator using `HAL_COMP_Init()` function:
 - Select the inverting input (COMP2 only)
 - Select the non-inverting input
 - Select the output redirection to timers (COMP2 only)
 - Select the speed mode (COMP2 only)
 - Select the window mode (related to COMP1 and COMP2, but selected by COMP2 only)
 - Select the pull-up/down resistors on non-inverting input (COMP1 only)
 3. Enable the comparator using `HAL_COMP_Start()` or `HAL_COMP_Start_IT()` function
 4. If needed, use `HAL_COMP_GetOutputLevel()` or `HAL_COMP_TriggerCallback()` functions to manage comparator actions (output level or events)
 5. Disable the comparator using `HAL_COMP_Stop()` or `HAL_COMP_Stop_IT()` function
 6. De-initialize the comparator using `HAL_COMP_DeInit()` function

6.3.3 Initialization and de-initialization functions

This section provides functions to initialize and de-initialize comparators

This section contains the following APIs:

- [`HAL_COMP_Init\(\)`](#)
- [`HAL_COMP_DeInit\(\)`](#)
- [`HAL_COMP_MspInit\(\)`](#)
- [`HAL_COMP_MspDeInit\(\)`](#)

6.3.4 IO operation functions

This subsection provides a set of functions allowing to manage the COMP start and stop actions with or without interruption on ExtI line.

This section contains the following APIs:

- [`HAL_COMP_Start\(\)`](#)
- [`HAL_COMP_Stop\(\)`](#)
- [`HAL_COMP_Start_IT\(\)`](#)
- [`HAL_COMP_Stop_IT\(\)`](#)
- [`HAL_COMP_IRQHandler\(\)`](#)

6.3.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the COMP management functions: Lock status, comparator output level check, IRQ callback (in case of usage of comparator with interruption on ExtI line).

This section contains the following APIs:

- [`HAL_COMP_Lock\(\)`](#)
- [`HAL_COMP_GetOutputLevel\(\)`](#)
- [`HAL_COMP_TriggerCallback\(\)`](#)

6.3.6 Peripheral State functions

This subsection permit to get in run-time the status of the peripheral.

This section contains the following APIs:

- [HAL_COMP_GetState\(\)](#)

6.3.7 HAL_COMP_Init

Function Name	HAL_StatusTypeDef HAL_COMP_Init (COMP_HandleTypeDef * hcomp)
Function Description	Initializes the COMP according to the specified parameters in the COMP_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hcomp: COMP handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • If the selected comparator is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

6.3.8 HAL_COMP_DeInit

Function Name	HAL_StatusTypeDef HAL_COMP_DeInit (COMP_HandleTypeDef * hcomp)
Function Description	DeInitializes the COMP peripheral.
Parameters	<ul style="list-style-type: none"> • hcomp: COMP handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • Deinitialization can't be performed if the COMP configuration is locked. To unlock the configuration, perform a system reset.

6.3.9 HAL_COMP_Msplnit

Function Name	void HAL_COMP_Msplnit (COMP_HandleTypeDef * hcomp)
Function Description	Initializes the COMP MSP.
Parameters	<ul style="list-style-type: none"> • hcomp: COMP handle
Return values	<ul style="list-style-type: none"> • None

6.3.10 HAL_COMP_MspDeInit

Function Name	void HAL_COMP_MspDeInit (COMP_HandleTypeDef * hcomp)
Function Description	DeInitializes COMP MSP.
Parameters	<ul style="list-style-type: none"> • hcomp: COMP handle
Return values	<ul style="list-style-type: none"> • None

6.3.11 HAL_COMP_Start

Function Name	HAL_StatusTypeDef HAL_COMP_Start
---------------	---

(COMP_HandleTypeDef * hcomp)

Function Description Start the comparator.

Parameters • **hcomp**: COMP handle

Return values • HAL status

6.3.12 HAL_COMP_Stop

Function Name **HAL_StatusTypeDef HAL_COMP_Stop**
(COMP_HandleTypeDef * hcomp)

Function Description Stop the comparator.

Parameters • **hcomp**: COMP handle

Return values • HAL status

6.3.13 HAL_COMP_Start_IT

Function Name **HAL_StatusTypeDef HAL_COMP_Start_IT**
(COMP_HandleTypeDef * hcomp)

Function Description Enables the interrupt and starts the comparator.

Parameters • **hcomp**: COMP handle

Return values • HAL status.

6.3.14 HAL_COMP_Stop_IT

Function Name **HAL_StatusTypeDef HAL_COMP_Stop_IT**
(COMP_HandleTypeDef * hcomp)

Function Description Disable the interrupt and Stop the comparator.

Parameters • **hcomp**: COMP handle

Return values • HAL status

6.3.15 HAL_COMP_IRQHandler

Function Name **void HAL_COMP_IRQHandler** (COMP_HandleTypeDef *
hcomp)

Function Description Comparator IRQ Handler.

Parameters • **hcomp**: COMP handle

Return values • HAL status

6.3.16 HAL_COMP_Lock

Function Name **HAL_StatusTypeDef HAL_COMP_Lock**
(COMP_HandleTypeDef * hcomp)

Function Description Lock the selected comparator configuration.

Parameters • **hcomp**: COMP handle

Return values • HAL status

6.3.17 HAL_COMP_GetOutputLevel

Function Name **uint32_t HAL_COMP_GetOutputLevel (COMP_HandleTypeDef * hcomp)**

Function Description Return the output level (high or low) of the selected comparator.

6.3.18 HAL_COMP_TriggerCallback

Function Name **void HAL_COMP_TriggerCallback (COMP_HandleTypeDef * hcomp)**

Function Description Comparator callback.

Parameters • **hcomp:** COMP handle

Return values • None

6.3.19 HAL_COMP_GetState

Function Name **HAL_COMP_StateTypeDef HAL_COMP_GetState (COMP_HandleTypeDef * hcomp)**

Function Description Return the COMP state.

Parameters • **hcomp:** : COMP handle

Return values • HAL state

6.4 COMP Firmware driver defines**6.4.1 COMP****COMP Exported Macro**

__HAL_COMP_RESET_HANDLE_STATE

Description:

- Reset COMP handle state.

Parameters:

- **__HANDLE__:** COMP handle.

Return value:

- None

__HAL_COMP_ENABLE

Description:

- Enables the specified comparator.

Parameters:

- **__HANDLE__:** COMP handle.

Return value:

- None.

__HAL_COMP_DISABLE

Description:

__HAL_COMP_GET_FLAG

- Disables the specified comparator.

Parameters:

- __HANDLE__: COMP handle.

Return value:

- None.

Description:

- Checks whether the specified COMP flag is set or not.

Parameters:

- __HANDLE__: specifies the COMP Handle.
- __FLAG__: specifies the flag to check. This parameter can be one of the following values:
 - COMP_FLAG_LOCK: lock flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_COMP_COMP1_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP1 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP1 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP1 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP1 EXTI line falling edge trigger.

__HAL_COMP_COMP1_EXTI_ENABLE_RISING_FALLING_EDGE

Return value:

- None

Description:

- Enable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable the COMP1 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_IT

Description:

- Enable the COMP1 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_IT

Description:

- Disable the COMP1 EXTI line in interrupt mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_ENABLE_EVENT

Description:

- Enable the COMP1 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_DISABLE_EVENT

Description:

- Disable the COMP1 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP1_EXTI_GET_FLAG

Description:

- Check whether the COMP1 EXTI line flag is set or not.

Return value:

__HAL_COMP_COMP1_EXTI_CLEAR_FLAG

- RESET: or SET

Description:

- Clear the the COMP1 EXTI flag.

Return value:

- None

__HAL_COMP_COMP1_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on COMP1 EXTI Line.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_RISING_EDGE

Description:

- Enable the COMP2 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_RISING_EDGE

Description:

- Disable the COMP2 EXTI line rising edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable the COMP2 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable the COMP2 EXTI line falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable the COMP2 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

NG_EDGE

- Disable the COMP2 EXTI line rising & falling edge trigger.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_IT

Description:

- Enable the COMP2 EXTI line.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_IT

Description:

- Disable the COMP2 EXTI line.

Return value:

- None

__HAL_COMP_COMP2_EXTI_ENABLE_EVENT

Description:

- Enable the COMP2 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_DISABLE_EVENT

Description:

- Disable the COMP2 EXTI Line in event mode.

Return value:

- None

__HAL_COMP_COMP2_EXTI_GET_FLAG

Description:

- Check whether the COMP2 EXTI line flag is set or not.

Return value:

- RESET: or SET

__HAL_COMP_COMP2_EXTI_CLEAR_FLAG

Description:

- Clear the COMP2 EXTI flag.

Return value:

- None

__HAL_COMP_COMP2_EXTI_GENERATE_SWIT

Description:

- Generates a Software interrupt on COMP1 EXTI Line.

Return value:

- None

COMP ExtiLineEvent

COMP_EXTI_LINE_COMP1 External interrupt line 21 Connected to COMP1

COMP_EXTI_LINE_COMP2 External interrupt line 22 Connected to COMP2

COMP InvertingInput

COMP_INVERTINGINPUT_IO External I/O (COMP2_INM connected to pin PB3) connected to comparator 2 inverting input

COMP_INVERTINGINPUT_VREFINT VREFINT connected to comparator 2 inverting input

COMP_INVERTINGINPUT_3_4VREFINT 3/4 VREFINT connected to comparator 2 inverting input

COMP_INVERTINGINPUT_1_2VREFINT 1/2 VREFINT connected to comparator 2 inverting input

COMP_INVERTINGINPUT_1_4VREFINT 1/4 VREFINT connected to comparator 2 inverting input

COMP_INVERTINGINPUT_DAC1 DAC_OUT1 (PA4) connected to comparator 2 inverting input

COMP_INVERTINGINPUT_DAC2 DAC2_OUT (PA5) connected to comparator 2 inverting input

IS_COMP_INVERTINGINPUT

COMP Mode

COMP_MODE_LOWSPEED Low Speed

COMP_MODE_HIGHSPEED High Speed

IS_COMP_MODE

COMP NonInvertingInputPull

COMP_NONINVERTINGINPUT_NOPULL No internal pull-up or pull-down resistor connected to comparator non inverting input

COMP_NONINVERTINGINPUT_10KPU Internal 10kOhm pull-up resistor connected to comparator non inverting input

COMP_NONINVERTINGINPUT_10KPD Internal 10kOhm pull-down resistor connected to comparator non inverting input

COMP_NONINVERTINGINPUT_400KPU Internal 400kOhm pull-up resistor connected to comparator non inverting input

COMP_NONINVERTINGINPUT_400KPD Internal 400kOhm pull-down resistor connected to comparator non inverting input

IS_COMP_NONINVERTINGINPUTPULL

COMP Output

COMP_OUTPUT_TIM2IC4 COMP2 output connected to TIM2 Input Capture 4

COMP_OUTPUT_TIM2OCREFCLR COMP2 output connected to TIM2 OCREF Clear

COMP_OUTPUT_TIM3IC4	COMP2 output connected to TIM3 Input Capture 4
COMP_OUTPUT_TIM3OCREFCLR	COMP2 output connected to TIM3 OCREF Clear
COMP_OUTPUT_TIM4IC4	COMP2 output connected to TIM4 Input Capture 4
COMP_OUTPUT_TIM4OCREFCLR	COMP2 output connected to TIM4 OCREF Clear
COMP_OUTPUT_TIM10IC1	COMP2 output connected to TIM10 Input Capture 1
COMP_OUTPUT_NONE	COMP2 output is not connected to other peripherals
IS_COMP_OUTPUT	

COMP OutputLevel

COMP_OUTPUTLEVEL_LOW

COMP_OUTPUTLEVEL_HIGH

COMP Private Constants

COMP1_START_DELAY_CPU_CYCLES

COMP2_START_DELAY_CPU_CYCLES

COMP_STATE_BIT_LOCK

COMP Private Macro

COMP_GET_EXTI_LINE

Description:

- Get the specified EXTI line for a comparator instance.

Parameters:

- `__INSTANCE__`: specifies the COMP instance.

Return value:

- value: of

__COMP_CSR_CMPXOUT

Description:

- Select the COMP register CSR bit CMPxOUT corresponding to the selected COMP instance.

Parameters:

- `__HANDLE__`: COMP handle

Return value:

- Comparator: register CSR bit COMP_CSR_CMP1OUT or COMP_CSR_CMP2OUT

__COMP_IS_ENABLED

Description:

- Verification of COMP state: enabled or disabled.

Parameters:

- `__HANDLE__`: COMP handle

Return value:

- SET: (COMP enabled) or RESET (COMP disabled)

COMP TriggerMode

COMP_TRIGGERMODE_NONE

No External Interrupt trigger detection

COMP_TRIGGERMODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
COMP_TRIGGERMODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
COMP_TRIGGERMODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
IS_COMP_TRIGGERMODE	
COMP WindowMode	
COMP_WINDOWMODE_DISABLE	Window mode disabled: COMP1 non-inverting input is independant
COMP_WINDOWMODE_ENABLE	Window mode enabled: COMP1 non-inverting input is no more accessible, even from ADC channel VCOMP) (connected to COMP2 non-inverting input)
IS_COMP_WINDOWMODE	

7 HAL COMP Extension Driver

7.1 HAL COMP Extension Driver

7.2 COMPEX Firmware driver defines

7.2.1 COMPEX

COMPEX NonInvertingInput

COMP_NONINVERTINGINPUT_PB4	I/O pin PB4 connection to COMP2 non-inverting input
COMP_NONINVERTINGINPUT_PB5	I/O pin PB5 connection to COMP2 non-inverting input
COMP_NONINVERTINGINPUT_PB6	I/O pin PB6 connection to COMP2 non-inverting input
COMP_NONINVERTINGINPUT_PB7	I/O pin PB7 connection to COMP2 non-inverting input
COMP_NONINVERTINGINPUT_NONE	In case of window mode: No I/O pin connection to COMP1 non-inverting input. Instead, connection to COMP2 non-inverting input.
COMP_NONINVERTINGINPUT_PA0	I/O pin PA0 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PA1	I/O pin PA1 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PA2	I/O pin PA2 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PA3	I/O pin PA3 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PA4	I/O pin PA4 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PA5	I/O pin PA5 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PA6	I/O pin PA5 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PA7	I/O pin PA7 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PB0	I/O pin PB0 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PB1	I/O pin PB1 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PC0	I/O pin PC0 connection to COMP1 non-inverting input

COMP_NONINVERTINGINPUT_PC1	I/O pin PC1 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PC2	I/O pin PC2 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PC3	I/O pin PC3 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PC4	I/O pin PC4 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PC5	I/O pin PC5 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PB12	I/O pin PB12 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PB13	I/O pin PB13 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PB14	I/O pin PB14 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PB15	I/O pin PB15 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PE7	I/O pin PE7 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PE8	I/O pin PE8 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PE9	I/O pin PE9 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PE10	I/O pin PE10 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PF6	I/O pin PF6 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PF7	I/O pin PF7 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PF8	I/O pin PF8 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PF9	I/O pin PF9 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_PF10	I/O pin PF10 connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_OPAMP1	OPAMP1 output connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_OPAMP2	OPAMP2 output connection to COMP1 non-inverting input
COMP_NONINVERTINGINPUT_OPAMP3	OPAMP3 output connection to COMP1 non-inverting input
IS_COMP_NONINVERTINGINPUT	

COMP Private Macro

`__COMP_ROUTING_INTERFACE_TOBECONFIGURED`**Description:**

- Specifies whether Routing Interface (RI) needs to be configured for switches of comparator non-inverting input.

Parameters:

- `__HANDLE__`: COMP handle.

Return value:

- None.

8 HAL CORTEX Generic Driver

8.1 HAL CORTEX Generic Driver

8.2 CORTEX Firmware driver registers structures

8.2.1 MPU_Region_InitTypeDef

Data Fields

- *uint8_t Enable*
- *uint8_t Number*
- *uint32_t BaseAddress*
- *uint8_t Size*
- *uint8_t SubRegionDisable*
- *uint8_t TypeExtField*
- *uint8_t AccessPermission*
- *uint8_t DisableExec*
- *uint8_t IsShareable*
- *uint8_t IsCacheable*
- *uint8_t IsBufferable*

Field Documentation

- ***uint8_t MPU_Region_InitTypeDef::Enable***
Specifies the status of the region. This parameter can be a value of [CORTEX_MPU_Region_Enable](#)
- ***uint8_t MPU_Region_InitTypeDef::Number***
Specifies the number of the region to protect. This parameter can be a value of [CORTEX_MPU_Region_Number](#)
- ***uint32_t MPU_Region_InitTypeDef::BaseAddress***
Specifies the base address of the region to protect.
- ***uint8_t MPU_Region_InitTypeDef::Size***
Specifies the size of the region to protect. This parameter can be a value of [CORTEX_MPU_Region_Size](#)
- ***uint8_t MPU_Region_InitTypeDef::SubRegionDisable***
Specifies the number of the subregion protection to disable. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint8_t MPU_Region_InitTypeDef::TypeExtField***
Specifies the TEX field level. This parameter can be a value of [CORTEX_MPU_TEX_Levels](#)
- ***uint8_t MPU_Region_InitTypeDef::AccessPermission***
Specifies the region access permission type. This parameter can be a value of [CORTEX_MPU_Region_Permission_Attributes](#)
- ***uint8_t MPU_Region_InitTypeDef::DisableExec***
Specifies the instruction access status. This parameter can be a value of [CORTEX_MPU_Instruction_Access](#)
- ***uint8_t MPU_Region_InitTypeDef::IsShareable***
Specifies the shareability status of the protected region. This parameter can be a value of [CORTEX_MPU_Access_Shareable](#)

- **`uint8_t MPU_Region_InitTypeDef::IsCacheable`**
Specifies the cacheable status of the region protected. This parameter can be a value of [CORTEX_MPU_Access_Cacheable](#)
- **`uint8_t MPU_Region_InitTypeDef::IsBufferable`**
Specifies the bufferable status of the protected region. This parameter can be a value of [CORTEX_MPU_Access_Bufferable](#)

8.3 CORTEX Firmware driver API description

8.3.1 Initialization and de-initialization functions

This section provide the Cortex HAL driver functions allowing to configure Interrupts SysTick functionalities

This section contains the following APIs:

- [HAL_NVIC_SetPriorityGrouping\(\)](#)
- [HAL_NVIC_SetPriority\(\)](#)
- [HAL_NVIC_EnableIRQ\(\)](#)
- [HAL_NVIC_DisableIRQ\(\)](#)
- [HAL_NVIC_SystemReset\(\)](#)
- [HAL_SYSTICK_Config\(\)](#)

8.3.2 Peripheral Control functions

This subsection provides a set of functions allowing to control the CORTEX (NVIC, SYSTICK, MPU) functionalities.

This section contains the following APIs:

- [HAL_MPU_ConfigRegion\(\)](#)
- [HAL_NVIC_GetPriorityGrouping\(\)](#)
- [HAL_NVIC_GetPriority\(\)](#)
- [HAL_NVIC_SetPendingIRQ\(\)](#)
- [HAL_NVIC_GetPendingIRQ\(\)](#)
- [HAL_NVIC_ClearPendingIRQ\(\)](#)
- [HAL_NVIC_GetActive\(\)](#)
- [HAL_SYSTICK_CLKSourceConfig\(\)](#)
- [HAL_SYSTICK_IRQHandler\(\)](#)
- [HAL_SYSTICK_Callback\(\)](#)

8.3.3 HAL_NVIC_SetPriorityGrouping

Function Name	<code>void HAL_NVIC_SetPriorityGrouping (uint32_t PriorityGroup)</code>
Function Description	Sets the priority grouping field (pre-emption priority and subpriority) using the required unlock sequence.
Parameters	<ul style="list-style-type: none"> • PriorityGroup: The priority grouping bits length. This parameter can be one of the following values: NVIC_PRIORITYGROUP_0: 0 bits for pre-emption priority 4 bits for subpriority NVIC_PRIORITYGROUP_1: 1 bits for pre-emption priority 3 bits for subpriority NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority 2 bits for subpriority NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority 1 bits for subpriority NVIC_PRIORITYGROUP_4: 4 bits for pre-emption

priority 0 bits for subpriority

- | | |
|---------------|---|
| Return values | <ul style="list-style-type: none"> • None |
| Notes | <ul style="list-style-type: none"> • When the NVIC_PriorityGroup_0 is selected, IRQ pre-emption is no more possible. The pending IRQ priority will be managed only by the subpriority. |

8.3.4 HAL_NVIC_SetPriority

Function Name **void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)**

Function Description Sets the priority of an interrupt.

- | | |
|------------|--|
| Parameters | <ul style="list-style-type: none"> • IRQn: External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l1xx.h)) • PreemptPriority: The pre-emption priority for the IRQn channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority • SubPriority: the subpriority level for the IRQ channel. This parameter can be a value between 0 and 15 A lower priority value indicates a higher priority. |
|------------|--|

- | | |
|---------------|--|
| Return values | <ul style="list-style-type: none"> • None |
|---------------|--|

8.3.5 HAL_NVIC_EnableIRQ

Function Name **void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)**

Function Description Enables a device specific interrupt in the NVIC interrupt controller.

- | | |
|------------|---|
| Parameters | <ul style="list-style-type: none"> • IRQn: External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l1xx.h)) |
|------------|---|

- | | |
|---------------|--|
| Return values | <ul style="list-style-type: none"> • None |
|---------------|--|

- | | |
|-------|--|
| Notes | <ul style="list-style-type: none"> • To configure interrupts priority correctly, the NVIC_PriorityGroupConfig() function should be called before. |
|-------|--|

8.3.6 HAL_NVIC_DisableIRQ

Function Name **void HAL_NVIC_DisableIRQ (IRQn_Type IRQn)**

Function Description Disables a device specific interrupt in the NVIC interrupt controller.

- | | |
|------------|--|
| Parameters | <ul style="list-style-type: none"> • IRQn: External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l1xxx.h)) |
|------------|--|

- | | |
|---------------|--|
| Return values | <ul style="list-style-type: none"> • None |
|---------------|--|

8.3.7 HAL_NVIC_SystemReset

Function Name **void HAL_NVIC_SystemReset (void)**

Function Description Initiates a system reset request to reset the MCU.

Return values

- None

8.3.8 HAL_SYSTICK_Config

Function Name **uint32_t HAL_SYSTICK_Config (uint32_t TicksNumb)**

Function Description Initializes the System Timer and its interrupt, and starts the System Tick Timer.

Parameters

- **TicksNumb:** Specifies the ticks Number of ticks between two interrupts.

Return values

- status - 0 Function succeeded. 1 Function failed.

8.3.9 HAL_MPU_ConfigRegion

Function Name **void HAL_MPU_ConfigRegion (MPU_Region_InitTypeDef * MPU_Init)**

Function Description Initializes and configures the Region and the memory to be protected.

Parameters

- **MPU_Init:** Pointer to a MPU_Region_InitTypeDef structure that contains the initialization and configuration information.

Return values

- None

8.3.10 HAL_NVIC_GetPriorityGrouping

Function Name **uint32_t HAL_NVIC_GetPriorityGrouping (void)**

Function Description Gets the priority grouping field from the NVIC Interrupt Controller.

Return values

- Priority grouping field (SCB->AIRCR [10:8] PRIGROUP field)

8.3.11 HAL_NVIC_GetPriority

Function Name **void HAL_NVIC_GetPriority (IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t * pPreemptPriority, uint32_t * pSubPriority)**

Function Description Gets the priority of an interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l1xxxx.h))
- **PriorityGroup:** the priority grouping bits length. This parameter can be one of the following values:
 NVIC_PRIORITYGROUP_0: 0 bits for pre-emption priority 4 bits for subpriority
 NVIC_PRIORITYGROUP_1: 1 bits for pre-emption priority 3 bits for subpriority
 NVIC_PRIORITYGROUP_2: 2 bits for pre-emption priority 2 bits for subpriority
 NVIC_PRIORITYGROUP_3: 3 bits for pre-emption priority 1 bits for subpriority
 NVIC_PRIORITYGROUP_4: 4 bits for pre-emption priority 0 bits for subpriority
- **pPreemptPriority:** Pointer on the Preemptive priority value

- (starting from 0).
- **pSubPriority:** Pointer on the Subpriority value (starting from 0).

Return values

- None

8.3.12 HAL_NVIC_SetPendingIRQ

Function Name **void HAL_NVIC_SetPendingIRQ (IRQn_Type IRQn)**

Function Description Sets Pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l1xxxx.h))

Return values

- None

8.3.13 HAL_NVIC_GetPendingIRQ

Function Name **uint32_t HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)**

Function Description Gets Pending Interrupt (reads the pending register in the NVIC and returns the pending bit for the specified interrupt).

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l1xxxx.h))

Return values

- status - 0 Interrupt status is not pending. 1 Interrupt status is pending.

8.3.14 HAL_NVIC_ClearPendingIRQ

Function Name **void HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)**

Function Description Clears the pending bit of an external interrupt.

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l1xxxx.h))

Return values

- None

8.3.15 HAL_NVIC_GetActive

Function Name **uint32_t HAL_NVIC_GetActive (IRQn_Type IRQn)**

Function Description Gets active interrupt (reads the active register in NVIC and returns the active bit).

Parameters

- **IRQn:** External interrupt number This parameter can be an enumerator of IRQn_Type enumeration (For the complete STM32 Devices IRQ Channels list, please refer to the appropriate CMSIS device file (stm32l1xxxx.h))

Return values

- status - 0 Interrupt status is not pending. 1 Interrupt status is

pending.

8.3.16 HAL_SYSTICK_CLKSourceConfig

Function Name	void HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)
Function Description	Configures the SysTick clock source.
Parameters	<ul style="list-style-type: none"> • CLKSource: specifies the SysTick clock source. This parameter can be one of the following values: SYSTICK_CLKSOURCE_HCLK_DIV8: AHB clock divided by 8 selected as SysTick clock source. SYSTICK_CLKSOURCE_HCLK: AHB clock selected as SysTick clock source.
Return values	<ul style="list-style-type: none"> • None

8.3.17 HAL_SYSTICK_IRQHandler

Function Name	void HAL_SYSTICK_IRQHandler (void)
Function Description	This function handles SYSTICK interrupt request.
Return values	<ul style="list-style-type: none"> • None

8.3.18 HAL_SYSTICK_Callback

Function Name	void HAL_SYSTICK_Callback (void)
Function Description	SYSTICK callback.
Return values	<ul style="list-style-type: none"> • None

8.4 CORTEX Firmware driver defines

8.4.1 CORTEX

CORTEX MPU Instruction Access Bufferable

MPU_ACCESS_BUFFERABLE

MPU_ACCESS_NOT_BUFFERABLE

CORTEX MPU Instruction Access Cacheable

MPU_ACCESS_CACHEABLE

MPU_ACCESS_NOT_CACHEABLE

CORTEX MPU Instruction Access Shareable

MPU_ACCESS_SHAREABLE

MPU_ACCESS_NOT_SHAREABLE

MPU HFNMI and PRIVILEGED Access control

MPU_HFNMI_PRIVDEF_NONE

MPU_HARDFAULT_NMI

MPU_PRIVILEGED_DEFAULT

MPU_HFNMI_PRIVDEF

CORTEX MPU Instruction Access

MPU_INSTRUCTION_ACCESS_ENABLE

MPU_INSTRUCTION_ACCESS_DISABLE

CORTEX MPU Region Enable

MPU_REGION_ENABLE

MPU_REGION_DISABLE

CORTEX MPU Region Number

MPU_REGION_NUMBER0

MPU_REGION_NUMBER1

MPU_REGION_NUMBER2

MPU_REGION_NUMBER3

MPU_REGION_NUMBER4

MPU_REGION_NUMBER5

MPU_REGION_NUMBER6

MPU_REGION_NUMBER7

CORTEX MPU Region Permission Attributes

MPU_REGION_NO_ACCESS

MPU_REGION_PRIV_RW

MPU_REGION_PRIV_RW_URO

MPU_REGION_FULL_ACCESS

MPU_REGION_PRIV_RO

MPU_REGION_PRIV_RO_URO

CORTEX MPU Region Size

MPU_REGION_SIZE_32B

MPU_REGION_SIZE_64B

MPU_REGION_SIZE_128B

MPU_REGION_SIZE_256B

MPU_REGION_SIZE_512B

MPU_REGION_SIZE_1KB

MPU_REGION_SIZE_2KB

MPU_REGION_SIZE_4KB

MPU_REGION_SIZE_8KB

MPU_REGION_SIZE_16KB

MPU_REGION_SIZE_32KB

MPU_REGION_SIZE_64KB

MPU_REGION_SIZE_128KB
MPU_REGION_SIZE_256KB
MPU_REGION_SIZE_512KB
MPU_REGION_SIZE_1MB
MPU_REGION_SIZE_2MB
MPU_REGION_SIZE_4MB
MPU_REGION_SIZE_8MB
MPU_REGION_SIZE_16MB
MPU_REGION_SIZE_32MB
MPU_REGION_SIZE_64MB
MPU_REGION_SIZE_128MB
MPU_REGION_SIZE_256MB
MPU_REGION_SIZE_512MB
MPU_REGION_SIZE_1GB
MPU_REGION_SIZE_2GB
MPU_REGION_SIZE_4GB

MPU TEX Levels

MPU_TEX_LEVEL0
MPU_TEX_LEVEL1
MPU_TEX_LEVEL2

CORTEX Preemption Priority Group

NVIC_PRIORITYGROUP_0 0 bits for pre-emption priority 4 bits for subpriority
NVIC_PRIORITYGROUP_1 1 bits for pre-emption priority 3 bits for subpriority
NVIC_PRIORITYGROUP_2 2 bits for pre-emption priority 2 bits for subpriority
NVIC_PRIORITYGROUP_3 3 bits for pre-emption priority 1 bits for subpriority
NVIC_PRIORITYGROUP_4 4 bits for pre-emption priority 0 bits for subpriority

CORTEX Preemption Priority Group

IS_NVIC_PRIORITY_GROUP
IS_NVIC_PREEMPTION_PRIORITY
IS_NVIC_SUB_PRIORITY
IS_NVIC_DEVICE_IRQ

CORTEX Private Macros

IS_MPU_REGION_ENABLE
IS_MPU_INSTRUCTION_ACCESS
IS_MPU_ACCESS_SHAREABLE
IS_MPU_ACCESS_CACHEABLE

IS_MPU_ACCESS_BUFFERABLE

IS_MPU_TEX_LEVEL

IS_MPU_REGION_PERMISSION_ATTRIBUTE

IS_MPU_REGION_NUMBER

IS_MPU_REGION_SIZE

IS_MPU_SUB_REGION_DISABLE

CORTEX SysTick clock source

SYSTICK_CLKSOURCE_HCLK_DIV8

SYSTICK_CLKSOURCE_HCLK

CORTEX SysTick clock source

__HAL_CORTEX_SYSTICKCLK_CONFIG

Description:

- Configures the SysTick clock source.

Parameters:

- `__CLKSRC__`: specifies the SysTick clock source. This parameter can be one of the following values:
 - `SYSTICK_CLKSOURCE_HCLK_DIV8`: AHB clock divided by 8 selected as SysTick clock source.
 - `SYSTICK_CLKSOURCE_HCLK`: AHB clock selected as SysTick clock source.

Return value:

- None

CORTEX SysTick clock source

IS_SYSTICK_CLK_SOURCE

9 HAL CRC Generic Driver

9.1 HAL CRC Generic Driver

9.2 CRC Firmware driver registers structures

9.2.1 CRC_HandleTypeDef

Data Fields

- *CRC_TypeDef * Instance*
- *HAL_LockTypeDef Lock*
- *__IO HAL_CRC_StateTypeDef State*

Field Documentation

- *CRC_TypeDef* CRC_HandleTypeDef::Instance*
Register base address
- *HAL_LockTypeDef CRC_HandleTypeDef::Lock*
CRC locking object
- *__IO HAL_CRC_StateTypeDef CRC_HandleTypeDef::State*
CRC communication state

9.3 CRC Firmware driver API description

9.3.1 How to use this driver

The CRC HAL driver can be used as follows:

1. Enable CRC AHB clock using `__HAL_RCC_CRC_CLK_ENABLE()`;
2. Use `HAL_CRC_Accumulate()` function to compute the CRC value of a 32-bit data buffer using combination of the previous CRC value and the new one.
3. Use `HAL_CRC_Calculate()` function to compute the CRC Value of a new 32-bit data buffer. This function resets the CRC computation unit before starting the computation to avoid getting wrong CRC values.

9.3.2 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRC according to the specified parameters in the `CRC_InitTypeDef` and create the associated handle
- DeInitialize the CRC peripheral
- Initialize the CRC MSP
- DeInitialize CRC MSP

This section contains the following APIs:

- [*HAL_CRC_Init\(\)*](#)
- [*HAL_CRC_DeInit\(\)*](#)
- [*HAL_CRC_MspInit\(\)*](#)

- [HAL_CRC_MspDeInit\(\)](#)

9.3.3 Peripheral Control functions

This section provides functions allowing to:

- Compute the 32-bit CRC value of 32-bit data buffer, using combination of the previous CRC value and the new one.
- Compute the 32-bit CRC value of 32-bit data buffer, independently of the previous CRC value.

This section contains the following APIs:

- [HAL_CRC_Accumulate\(\)](#)
- [HAL_CRC_Calculate\(\)](#)

9.3.4 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- [HAL_CRC_GetState\(\)](#)
- [HAL_CRC_Accumulate\(\)](#)
- [HAL_CRC_Calculate\(\)](#)

9.3.5 HAL_CRC_Init

Function Name	HAL_StatusTypeDef HAL_CRC_Init (CRC_HandleTypeDef * hcrc)
Function Description	Initializes the CRC according to the specified parameters in the CRC_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hcrc: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	<ul style="list-style-type: none"> • HAL status

9.3.6 HAL_CRC_DeInit

Function Name	HAL_StatusTypeDef HAL_CRC_DeInit (CRC_HandleTypeDef * hcrc)
Function Description	DeInitializes the CRC peripheral.
Parameters	<ul style="list-style-type: none"> • hcrc: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	<ul style="list-style-type: none"> • HAL status

9.3.7 HAL_CRC_MspInit

Function Name	void HAL_CRC_MspInit (CRC_HandleTypeDef * hcrc)
Function Description	Initializes the CRC MSP.
Parameters	<ul style="list-style-type: none"> • hcrc: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	<ul style="list-style-type: none"> • None

9.3.8 HAL_CRC_MspDeInit

Function Name	void HAL_CRC_MspDeInit (CRC_HandleTypeDef * hcrc)
Function Description	DeInitializes the CRC MSP.
Parameters	<ul style="list-style-type: none">• hcrc: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	<ul style="list-style-type: none">• None

9.3.9 HAL_CRC_Accumulate

Function Name	uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)
Function Description	Computes the 32-bit CRC of 32-bit data buffer using combination of the previous CRC value and the new one.
Parameters	<ul style="list-style-type: none">• hcrc: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC• pBuffer: pointer to the buffer containing the data to be computed• BufferLength: length of the buffer to be computed (defined in word, 4 bytes)
Return values	<ul style="list-style-type: none">• 32-bit CRC

9.3.10 HAL_CRC_Calculate

Function Name	uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)
Function Description	Computes the 32-bit CRC of 32-bit data buffer independently of the previous CRC value.
Parameters	<ul style="list-style-type: none">• hcrc: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC• pBuffer: Pointer to the buffer containing the data to be computed• BufferLength: Length of the buffer to be computed (defined in word, 4 bytes)
Return values	<ul style="list-style-type: none">• 32-bit CRC

9.3.11 HAL_CRC_GetState

Function Name	HAL_CRC_StateTypeDef HAL_CRC_GetState (CRC_HandleTypeDef * hcrc)
Function Description	Returns the CRC state.
Parameters	<ul style="list-style-type: none">• hcrc: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC
Return values	<ul style="list-style-type: none">• HAL state

9.3.12 HAL_CRC_Accumulate

Function Name	uint32_t HAL_CRC_Accumulate (CRC_HandleTypeDef * hcrc,
---------------	---

uint32_t pBuffer, uint32_t BufferLength)

Function Description	Computes the 32-bit CRC of 32-bit data buffer using combination of the previous CRC value and the new one.
Parameters	<ul style="list-style-type: none"> • hcrc: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC • pBuffer: pointer to the buffer containing the data to be computed • BufferLength: length of the buffer to be computed (defined in word, 4 bytes)
Return values	<ul style="list-style-type: none"> • 32-bit CRC

9.3.13 HAL_CRC_Calculate

Function Name	uint32_t HAL_CRC_Calculate (CRC_HandleTypeDef * hcrc, uint32_t pBuffer, uint32_t BufferLength)
Function Description	Computes the 32-bit CRC of 32-bit data buffer independently of the previous CRC value.
Parameters	<ul style="list-style-type: none"> • hcrc: pointer to a CRC_HandleTypeDef structure that contains the configuration information for CRC • pBuffer: Pointer to the buffer containing the data to be computed • BufferLength: Length of the buffer to be computed (defined in word, 4 bytes)
Return values	<ul style="list-style-type: none"> • 32-bit CRC

9.4 CRC Firmware driver defines

9.4.1 CRC

CRC Exported Macros

__HAL_CRC_RESET_HANDLE_STATE	Description: <ul style="list-style-type: none"> • Reset CRC handle state. Parameters: <ul style="list-style-type: none"> • __HANDLE__: CRC handle Return value: <ul style="list-style-type: none"> • None
__HAL_CRC_DR_RESET	Description: <ul style="list-style-type: none"> • Resets CRC Data Register. Parameters: <ul style="list-style-type: none"> • __HANDLE__: CRC handle Return value: <ul style="list-style-type: none"> • None
__HAL_CRC_SET_IDR	Description: <ul style="list-style-type: none"> • Stores a 8-bit data in the Independent

Data(ID) register.

Parameters:

- `__HANDLE__`: CRC handle
- `__VALUE__`: 8-bit value to be stored in the ID register

Return value:

- None

Description:

- Returns the 8-bit data stored in the Independent Data(ID) register.

Parameters:

- `__HANDLE__`: CRC handle

Return value:

- 8-bit: value of the ID register

`__HAL_CRC_GET_IDR`

10 HAL CRYPT Generic Driver

10.1 HAL CRYPT Generic Driver

10.2 CRYPT Firmware driver registers structures

10.2.1 CRYPT_InitTypeDef

Data Fields

- *uint32_t* **DataType**
- *uint8_t ** **pKey**
- *uint8_t ** **pInitVect**

Field Documentation

- *uint32_t* **CRYPT_InitTypeDef::DataType**
32-bit data, 16-bit data, 8-bit data or 1-bit string. This parameter can be a value of [CRYPT_Data_Type](#)
- *uint8_t ** **CRYPT_InitTypeDef::pKey**
The key used for encryption/decryption
- *uint8_t ** **CRYPT_InitTypeDef::pInitVect**
The initialization vector used also as initialization counter in CTR mode

10.2.2 CRYPT_HandleTypeDef

Data Fields

- *AES_TypeDef ** **Instance**
- *CRYPT_InitTypeDef* **Init**
- *uint8_t ** **pCrypInBuffPtr**
- *uint8_t ** **pCrypOutBuffPtr**
- *__IO uint16_t* **CrypInCount**
- *__IO uint16_t* **CrypOutCount**
- *HAL_StatusTypeDef* **Status**
- *HAL_PhaseTypeDef* **Phase**
- *DMA_HandleTypeDef ** **hdmain**
- *DMA_HandleTypeDef ** **hdmaout**
- *HAL_LockTypeDef* **Lock**
- *__IO HAL_CRYPT_STATETTypeDef* **State**

Field Documentation

- *AES_TypeDef ** **CRYPT_HandleTypeDef::Instance**
Register base address
- *CRYPT_InitTypeDef* **CRYPT_HandleTypeDef::Init**
CRYPT required parameters

- ***uint8_t* CRYPT_HandleTypeDef::pCrypInBuffPtr***
Pointer to CRYPT processing (encryption, decryption,...) buffer
- ***uint8_t* CRYPT_HandleTypeDef::pCrypOutBuffPtr***
Pointer to CRYPT processing (encryption, decryption,...) buffer
- ***_IO uint16_t CRYPT_HandleTypeDef::CrypInCount***
Counter of input data
- ***_IO uint16_t CRYPT_HandleTypeDef::CrypOutCount***
Counter of output data
- ***HAL_StatusTypeDef CRYPT_HandleTypeDef::Status***
CRYPT peripheral status
- ***HAL_PhaseTypeDef CRYPT_HandleTypeDef::Phase***
CRYPT peripheral phase
- ***DMA_HandleTypeDef* CRYPT_HandleTypeDef::hdmain***
CRYPT In DMA handle parameters
- ***DMA_HandleTypeDef* CRYPT_HandleTypeDef::hdmaout***
CRYPT Out DMA handle parameters
- ***HAL_LockTypeDef CRYPT_HandleTypeDef::Lock***
CRYPT locking object
- ***_IO HAL_CRYPT_STATTypeDef CRYPT_HandleTypeDef::State***
CRYPT peripheral state

10.3 CRYPT Firmware driver API description

10.3.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the CRYPT according to the specified parameters in the CRYPT_InitTypeDef and creates the associated handle
- DeInitialize the CRYPT peripheral
- Initialize the CRYPT MSP
- DeInitialize CRYPT MSP

This section contains the following APIs:

- [***HAL_CRYPT_Init\(\)***](#)
- [***HAL_CRYPT_DeInit\(\)***](#)
- [***HAL_CRYPT_MspInit\(\)***](#)
- [***HAL_CRYPT_MspDeInit\(\)***](#)

10.3.2 AES processing functions

This section provides functions allowing to:

- Encrypt plaintext using AES algorithm in different chaining modes
- Decrypt cyphertext using AES algorithm in different chaining modes

Three processing functions are available:

- Polling mode
- Interrupt mode
- DMA mode

This section contains the following APIs:

- [***HAL_CRYPT_AESECB_Encrypt\(\)***](#)
- [***HAL_CRYPT_AESCBC_Encrypt\(\)***](#)

- [HAL_CRYPT_AESCTR_Encrypt\(\)](#)
- [HAL_CRYPT_AESECB_Decrypt\(\)](#)
- [HAL_CRYPT_AESCBC_Decrypt\(\)](#)
- [HAL_CRYPT_AESCTR_Decrypt\(\)](#)
- [HAL_CRYPT_AESECB_Encrypt_IT\(\)](#)
- [HAL_CRYPT_AESCBC_Encrypt_IT\(\)](#)
- [HAL_CRYPT_AESCTR_Encrypt_IT\(\)](#)
- [HAL_CRYPT_AESECB_Decrypt_IT\(\)](#)
- [HAL_CRYPT_AESCBC_Decrypt_IT\(\)](#)
- [HAL_CRYPT_AESCTR_Decrypt_IT\(\)](#)
- [HAL_CRYPT_AESECB_Encrypt_DMA\(\)](#)
- [HAL_CRYPT_AESCBC_Encrypt_DMA\(\)](#)
- [HAL_CRYPT_AESCTR_Encrypt_DMA\(\)](#)
- [HAL_CRYPT_AESECB_Decrypt_DMA\(\)](#)
- [HAL_CRYPT_AESCBC_Decrypt_DMA\(\)](#)
- [HAL_CRYPT_AESCTR_Decrypt_DMA\(\)](#)

10.3.3 DMA callback functions

This section provides DMA callback functions:

- DMA Input data transfer complete
- DMA Output data transfer complete
- DMA error

This section contains the following APIs:

- [HAL_CRYPT_ErrorCallback\(\)](#)
- [HAL_CRYPT_InCpltCallback\(\)](#)
- [HAL_CRYPT_OutCpltCallback\(\)](#)

10.3.4 CRYPT IRQ handler management

This section provides CRYPT IRQ handler function.

This section contains the following APIs:

- [HAL_CRYPT_IRQHandler\(\)](#)

10.3.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral.

This section contains the following APIs:

- [HAL_CRYPT_GetState\(\)](#)

10.3.6 HAL_CRYPT_Init

Function Name	HAL_StatusTypeDef HAL_CRYPT_Init (CRYPT_HandleTypeDef * hcrypt)
Function Description	Initializes the CRYPT according to the specified parameters in the CRYPT_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	<ul style="list-style-type: none"> • HAL status

10.3.7 HAL_CRYPT_DeInit

Function Name	HAL_StatusTypeDef HAL_CRYPT_DeInit (CRYPT_HandleTypeDef * hcrypt)
Function Description	DeInitializes the CRYPT peripheral.
Parameters	<ul style="list-style-type: none"> hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	<ul style="list-style-type: none"> HAL status

10.3.8 HAL_CRYPT_MspInit

Function Name	void HAL_CRYPT_MspInit (CRYPT_HandleTypeDef * hcrypt)
Function Description	Initializes the CRYPT MSP.
Parameters	<ul style="list-style-type: none"> hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	<ul style="list-style-type: none"> None

10.3.9 HAL_CRYPT_MspDeInit

Function Name	void HAL_CRYPT_MspDeInit (CRYPT_HandleTypeDef * hcrypt)
Function Description	DeInitializes CRYPT MSP.
Parameters	<ul style="list-style-type: none"> hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	<ul style="list-style-type: none"> None

10.3.10 HAL_CRYPT_AESECB_Encrypt

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESECB_Encrypt (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES ECB encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module pPlainData: Pointer to the plaintext buffer (aligned on u32) Size: Length of the plaintext buffer, must be a multiple of 16. pCypherData: Pointer to the cyphertext buffer (aligned on u32) Timeout: Specify Timeout value
Return values	<ul style="list-style-type: none"> HAL status

10.3.11 HAL_CRYPT_AESCBC_Encrypt

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESCBC_Encrypt (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES CBC encryption mode then

encrypt pPlainData.

Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pPlainData: Pointer to the plaintext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16. • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Timeout: Specify Timeout value
Return values	<ul style="list-style-type: none"> • HAL status

10.3.12 HAL_CRYPT_AESCTR_Encrypt

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESCTR_Encrypt (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES CTR encryption mode then encrypt pPlainData.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pPlainData: Pointer to the plaintext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16. • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Timeout: Specify Timeout value
Return values	<ul style="list-style-type: none"> • HAL status

10.3.13 HAL_CRYPT_AESECBC_Decrypt

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESECBC_Decrypt (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES ECB decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16. • pPlainData: Pointer to the plaintext buffer (aligned on u32) • Timeout: Specify Timeout value
Return values	<ul style="list-style-type: none"> • HAL status

10.3.14 HAL_CRYPT_AESCBC_Decrypt

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESCBC_Decrypt (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES ECB decryption mode then decrypted pCypherData.

Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16. • pPlainData: Pointer to the plaintext buffer (aligned on u32) • Timeout: Specify Timeout value
Return values	<ul style="list-style-type: none"> • HAL status

10.3.15 HAL_CRYPT_AESCTR_Decrypt

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESCTR_Decrypt (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData, uint32_t Timeout)
Function Description	Initializes the CRYPT peripheral in AES CTR decryption mode then decrypted pCypherData.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16. • pPlainData: Pointer to the plaintext buffer (aligned on u32) • Timeout: Specify Timeout value
Return values	<ul style="list-style-type: none"> • HAL status

10.3.16 HAL_CRYPT_AESECBC_Encrypt_IT

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESECBC_Encrypt_IT (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in AES ECB encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pPlainData: Pointer to the plaintext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16 bytes • pCypherData: Pointer to the cyphertext buffer (aligned on u32)
Return values	<ul style="list-style-type: none"> • HAL status

10.3.17 HAL_CRYPT_AESCBC_Encrypt_IT

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESCBC_Encrypt_IT (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)
Function Description	Initializes the CRYPT peripheral in AES CBC encryption mode using Interrupt.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that

- contains the configuration information for CRYPT module
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
- Return values
- HAL status

10.3.18 HAL_CRYPT_AESCTR_Encrypt_IT

- Function Name** **HAL_StatusTypeDef HAL_CRYPT_AESCTR_Encrypt_IT (CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**
- Function Description** Initializes the CRYPT peripheral in AES CTR encryption mode using Interrupt.
- Parameters**
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
- Return values
- HAL status

10.3.19 HAL_CRYPT_AESECB_Decrypt_IT

- Function Name** **HAL_StatusTypeDef HAL_CRYPT_AESECB_Decrypt_IT (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
- Function Description** Initializes the CRYPT peripheral in AES ECB decryption mode using Interrupt.
- Parameters**
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16.
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
- Return values
- HAL status

10.3.20 HAL_CRYPT_AESCBC_Decrypt_IT

- Function Name** **HAL_StatusTypeDef HAL_CRYPT_AESCBC_Decrypt_IT (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**
- Function Description** Initializes the CRYPT peripheral in AES CBC decryption mode using IT.
- Parameters**
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)

- **Size:** Length of the plaintext buffer, must be a multiple of 16
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
- Return values
- HAL status

10.3.21 HAL_CRYPT_AESCTR_Decrypt_IT

Function Name HAL_StatusTypeDef HAL_CRYPT_AESCTR_Decrypt_IT
(CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)

Function Description Initializes the CRYPT peripheral in AES CTR decryption mode using Interrupt.

- Parameters**
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)

- Return values
- HAL status

10.3.22 HAL_CRYPT_AESECB_Encrypt_DMA

Function Name HAL_StatusTypeDef HAL_CRYPT_AESECB_Encrypt_DMA
(CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)

Function Description Initializes the CRYPT peripheral in AES ECB encryption mode using DMA.

- Parameters**
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16 bytes
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)

- Return values
- HAL status

10.3.23 HAL_CRYPT_AESCBC_Encrypt_DMA

Function Name HAL_StatusTypeDef HAL_CRYPT_AESCBC_Encrypt_DMA
(CRYPT_HandleTypeDef * hcrypt, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)

Function Description Initializes the CRYPT peripheral in AES CBC encryption mode using DMA.

- Parameters**
- **hcryp:** pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
 - **pPlainData:** Pointer to the plaintext buffer (aligned on u32)
 - **Size:** Length of the plaintext buffer, must be a multiple of 16.
 - **pCypherData:** Pointer to the cyphertext buffer (aligned on u32)

Return values

- HAL status

10.3.24 HAL_CRYPT_AESCTR_Encrypt_DMA

Function Name **HAL_StatusTypeDef HAL_CRYPT_AESCTR_Encrypt_DMA (CRYPT_HandleTypeDef * hcryp, uint8_t * pPlainData, uint16_t Size, uint8_t * pCypherData)**

Function Description Initializes the CRYPT peripheral in AES CTR encryption mode using DMA.

Parameters

- **hcryp**: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
- **pPlainData**: Pointer to the plaintext buffer (aligned on u32)
- **Size**: Length of the plaintext buffer, must be a multiple of 16.
- **pCypherData**: Pointer to the cyphertext buffer (aligned on u32)

Return values

- HAL status

10.3.25 HAL_CRYPT_AESECBC_Decrypt_DMA

Function Name **HAL_StatusTypeDef HAL_CRYPT_AESECBC_Decrypt_DMA (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**

Function Description Initializes the CRYPT peripheral in AES ECB decryption mode using DMA.

Parameters

- **hcryp**: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
- **pCypherData**: Pointer to the cyphertext buffer (aligned on u32)
- **Size**: Length of the plaintext buffer, must be a multiple of 16 bytes
- **pPlainData**: Pointer to the plaintext buffer (aligned on u32)

Return values

- HAL status

10.3.26 HAL_CRYPT_AESCBC_Decrypt_DMA

Function Name **HAL_StatusTypeDef HAL_CRYPT_AESCBC_Decrypt_DMA (CRYPT_HandleTypeDef * hcryp, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)**

Function Description Initializes the CRYPT peripheral in AES CBC encryption mode using DMA.

Parameters

- **hcryp**: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
- **pCypherData**: Pointer to the cyphertext buffer (aligned on u32)
- **Size**: Length of the plaintext buffer, must be a multiple of 16 bytes
- **pPlainData**: Pointer to the plaintext buffer (aligned on u32)

Return values

- HAL status

10.3.27 HAL_CRYPT_AESCTR_Decrypt_DMA

Function Name	HAL_StatusTypeDef HAL_CRYPT_AESCTR_Decrypt_DMA (CRYPT_HandleTypeDef * hcrypt, uint8_t * pCypherData, uint16_t Size, uint8_t * pPlainData)
Function Description	Initializes the CRYPT peripheral in AES CTR decryption mode using DMA.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module • pCypherData: Pointer to the cyphertext buffer (aligned on u32) • Size: Length of the plaintext buffer, must be a multiple of 16 • pPlainData: Pointer to the plaintext buffer (aligned on u32)
Return values	<ul style="list-style-type: none"> • HAL status

10.3.28 HAL_CRYPT_ErrorCallback

Function Name	void HAL_CRYPT_ErrorCallback (CRYPT_HandleTypeDef * hcrypt)
Function Description	CRYPT error callback.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	<ul style="list-style-type: none"> • None

10.3.29 HAL_CRYPT_InCpltCallback

Function Name	void HAL_CRYPT_InCpltCallback (CRYPT_HandleTypeDef * hcrypt)
Function Description	Input transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	<ul style="list-style-type: none"> • None

10.3.30 HAL_CRYPT_OutCpltCallback

Function Name	void HAL_CRYPT_OutCpltCallback (CRYPT_HandleTypeDef * hcrypt)
Function Description	Output transfer completed callback.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module
Return values	<ul style="list-style-type: none"> • None

10.3.31 HAL_CRYPT_IRQHandler

Function Name	void HAL_CRYPT_IRQHandler (CRYPT_HandleTypeDef * hcrypt)
Function Description	This function handles CRYPT interrupt request.
Parameters	<ul style="list-style-type: none"> • hcryp: pointer to a CRYPT_HandleTypeDef structure that

contains the configuration information for CRYPT module

Return values

- None

10.3.32 HAL_CRYPT_GetState

Function Name **HAL_CRYPT_STATETTypeDef HAL_CRYPT_GetState (CRYPT_HandleTypeDef * hcryp)**

Function Description Returns the CRYPT state.

Parameters

- **hcryp**: pointer to a CRYPT_HandleTypeDef structure that contains the configuration information for CRYPT module

Return values

- HAL state

10.4 CRYPT Firmware driver defines

10.4.1 CRYPT

AES Clear Flags

CRYPT_CLEARFLAG_CCF Computation Complete Flag Clear

CRYPT_CLEARFLAG_RDERR Read Error Clear

CRYPT_CLEARFLAG_WRERR Write Error Clear

AES Flags

CRYPT_FLAG_CCF Computation Complete Flag

CRYPT_FLAG_RDERR Read Error Flag

CRYPT_FLAG_WRERR Write Error Flag

AES Interrupts

CRYPT_IT_CC Computation Complete interrupt

CRYPT_IT_ERR Error interrupt

CRYPT Algo Mode Direction

CRYPT_CR_ALGOMODE_DIRECTION

CRYPT_CR_ALGOMODE_AES_ECB_ENCRYPT

CRYPT_CR_ALGOMODE_AES_ECB_KEYDERDECRYPT

CRYPT_CR_ALGOMODE_AES_CBC_ENCRYPT

CRYPT_CR_ALGOMODE_AES_CBC_KEYDERDECRYPT

CRYPT_CR_ALGOMODE_AES_CTR_ENCRYPT

CRYPT_CR_ALGOMODE_AES_CTR_DECRYPT

CRYPT Data Type

CRYPT_DATATYPE_32B

CRYPT_DATATYPE_16B

CRYPT_DATATYPE_8B

CRYPT_DATATYPE_1B

IS_CRYPT_DATATYPE

CRYPT Exported Macros

__HAL_CRYPT_RESET_HANDLE_STATE

Description:

- Reset CRYPT handle state.

Parameters:

- `__HANDLE__`: specifies the CRYPT handle.

Return value:

- None

__HAL_CRYPT_ENABLE

Description:

- Enable/Disable the CRYPT peripheral.

Parameters:

- `__HANDLE__`: specifies the CRYPT handle.

Return value:

- None

__HAL_CRYPT_DISABLE

__HAL_CRYPT_SET_MODE

Description:

- Set the algorithm mode: AES-ECB, AES-CBC, AES-CTR, DES-ECB, DES-CBC,...

Parameters:

- `__HANDLE__`: specifies the CRYPT handle.
- `__MODE__`: The algorithm mode.

Return value:

- None

__HAL_CRYPT_GET_FLAG

Description:

- Check whether the specified CRYPT flag is set or not.

Parameters:

- `__HANDLE__`: specifies the CRYPT handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - CRYPT_FLAG_CCF : Computation Complete Flag
 - CRYPT_FLAG_RDERR : Read Error Flag
 - CRYPT_FLAG_WRERR : Write Error Flag

__HAL_CRYPT_CLEAR_FLAG**Return value:**

- The: new state of __FLAG__ (TRUE or FALSE).

Description:

- Clear the CRYPT pending flag.

Parameters:

- __HANDLE__: specifies the CRYPT handle.
- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
 - CRYPT_CLEARFLAG_CCF : Computation Complete Clear Flag
 - CRYPT_CLEARFLAG_RDERR : Read Error Clear
 - CRYPT_CLEARFLAG_WRERR : Write Error Clear

Return value:

- None

Description:

- Enable the CRYPT interrupt.

Parameters:

- __HANDLE__: specifies the CRYPT handle.
- __INTERRUPT__: CRYPT Interrupt.

Return value:

- None

Description:

- Disable the CRYPT interrupt.

Parameters:

- __HANDLE__: specifies the CRYPT handle.
- __INTERRUPT__: CRYPT interrupt.

Return value:

- None

Description:

- Checks if the specified CRYPT interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the CRYPT handle.
- __INTERRUPT__: CRYPT interrupt source

__HAL_CRYPT_ENABLE_IT**__HAL_CRYPT_DISABLE_IT****__HAL_CRYPT_GET_IT_SOURCE**

`__HAL_CRYPT_CLEAR_IT`

to check This parameter can be one of the following values:

- CRYPT_IT_CC : Computation Complete interrupt
- CRYPT_IT_ERR : Error interrupt (used for RDERR and WRERR)

Return value:

- State: of interruption (SET or RESET)

Description:

- Clear the CRYPT pending IT.

Parameters:

- `__HANDLE__`: specifies the CRYPT handle.
- `__IT__`: specifies the IT to clear. This parameter can be one of the following values:
 - CRYPT_CLEARFLAG_CCF : Computation Complete Clear Flag
 - CRYPT_CLEARFLAG_RDERR : Read Error Clear
 - CRYPT_CLEARFLAG_WRERR : Write Error Clear

Return value:

- None

CRYPT Private Defines

`CRYPT_ALGO_CHAIN_MASK`

11 HAL CRYP Extension Driver

11.1 HAL CRYP Extension Driver

11.2 CRYPEX Firmware driver API description

11.2.1 Extended features functions

This section provides callback functions:

- Computation completed.

This section contains the following APIs:

- [*HAL_CRYPEX_ComputationCpltCallback\(\)*](#)

11.2.2 HAL_CRYPEX_ComputationCpltCallback

Function Name	void HAL_CRYPEX_ComputationCpltCallback (CRYP_HandleTypeDef * hcryp)
Function Description	Computation completed callbacks.
Parameters	<ul style="list-style-type: none">• hcryp: pointer to a CRYP_HandleTypeDef structure that contains the configuration information for CRYP module
Return values	<ul style="list-style-type: none">• None

11.3 CRYPEX Firmware driver defines

11.3.1 CRYPEX

12 HAL DAC Generic Driver

12.1 HAL DAC Generic Driver

12.2 DAC Firmware driver registers structures

12.2.1 DAC_HandleTypeDef

Data Fields

- *DAC_TypeDef * Instance*
- *__IO HAL_DAC_StateTypeDef State*
- *HAL_LockTypeDef Lock*
- *DMA_HandleTypeDef * DMA_Handle1*
- *DMA_HandleTypeDef * DMA_Handle2*
- *__IO uint32_t ErrorCode*

Field Documentation

- *DAC_TypeDef* DAC_HandleTypeDef::Instance*
Register base address
- *__IO HAL_DAC_StateTypeDef DAC_HandleTypeDef::State*
DAC communication state
- *HAL_LockTypeDef DAC_HandleTypeDef::Lock*
DAC locking object
- *DMA_HandleTypeDef* DAC_HandleTypeDef::DMA_Handle1*
Pointer DMA handler for channel 1
- *DMA_HandleTypeDef* DAC_HandleTypeDef::DMA_Handle2*
Pointer DMA handler for channel 2
- *__IO uint32_t DAC_HandleTypeDef::ErrorCode*
DAC Error code

12.2.2 DAC_ChannelConfTypeDef

Data Fields

- *uint32_t DAC_Trigger*
- *uint32_t DAC_OutputBuffer*

Field Documentation

- *uint32_t DAC_ChannelConfTypeDef::DAC_Trigger*
Specifies the external trigger for the selected DAC channel. This parameter can be a value of [DAC_trigger_selection](#)
- *uint32_t DAC_ChannelConfTypeDef::DAC_OutputBuffer*
Specifies whether the DAC channel output buffer is enabled or disabled. This parameter can be a value of [DAC_output_buffer](#)

12.3 DAC Firmware driver API description

12.3.1 DAC Peripheral features

DAC Channels

The device integrates two 12-bit Digital Analog Converters that can be used independently or simultaneously (dual mode):

1. DAC channel1 with DAC_OUT1 (PA4) as output
2. DAC channel2 with DAC_OUT2 (PA5) as output

DAC Triggers

Digital to Analog conversion can be non-triggered using DAC_TRIGGER_NONE and DAC_OUT1/DAC_OUT2 is available once writing to DHRx register.

Digital to Analog conversion can be triggered by:

1. External event: EXTI Line 9 (any GPIOx_PIN_9) using DAC_TRIGGER_EXT_IT9.
The used pin (GPIOx_PIN_9) must be configured in input mode.
2. Timers TRGO: TIM2, TIM4, TIM6, TIM7, TIM9 (DAC_Trigger_T2_TRGO, DAC_Trigger_T4_TRGO...)
3. Software using DAC_TRIGGER_SOFTWARE

DAC Buffer mode feature

Each DAC channel integrates an output buffer that can be used to reduce the output impedance, and to drive external loads directly without having to add an external operational amplifier. To enable, the output buffer use `sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE`;



Refer to the device datasheet for more details about output impedance value with and without output buffer.

DAC connect feature

Each DAC channel can be connected internally. To connect, use `sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_ENABLE`;

GPIO configurations guidelines

When a DAC channel is used (ex channel1 on PA4) and the other is not (ex channel1 on PA5 is configured in Analog and disabled). Channel1 may disturb channel2 as coupling effect. Note that there is no coupling on channel2 as soon as channel2 is turned on. Coupling on adjacent channel could be avoided as follows: when unused PA5 is configured as INPUT PULL-UP or DOWN. PA5 is configured in ANALOG just before it is turned on.

DAC wave generation feature

Both DAC channels can be used to generate

1. Noise wave using `HAL_DACEx_NoiseWaveGenerate()`

2. Triangle wave using HAL_DACEx_TriangleWaveGenerate()

DAC data format

The DAC data format can be:

1. 8-bit right alignment using DAC_ALIGN_8B_R
2. 12-bit left alignment using DAC_ALIGN_12B_L
3. 12-bit right alignment using DAC_ALIGN_12B_R

DAC data value to voltage correspondance

The analog output voltage on each DAC channel pin is determined by the following equation:

$$\text{DAC_OUTx} = \text{VREF+} * \text{DOR} / 4095$$

- with DOR is the Data Output Register

VEF+ is the input voltage reference (refer to the device datasheet)

e.g. To set DAC_OUT1 to 0.7V, use

- Assuming that VREF+ = 3.3V, $\text{DAC_OUT1} = (3.3 * 868) / 4095 = 0.7\text{V}$

DMA requests

A DMA1 request can be generated when an external trigger (but not a software trigger) occurs if DMA1 requests are enabled using HAL_DAC_Start_DMA()

DMA1 requests are mapped as following:

1. DAC channel1 : mapped on DMA1 channel2 which must be already configured
2. DAC channel2 : mapped on DMA1 channel3 which must be already configured For Dual mode and specific signal (Triangle and noise) generation please refer to Extension Features Driver description

12.3.2 How to use this driver

- DAC APB clock must be enabled to get write access to DAC registers using HAL_DAC_Init()
- Configure DAC_OUTx (DAC_OUT1: PA4, DAC_OUT2: PA5) in analog mode.
- Configure the DAC channel using HAL_DAC_ConfigChannel() function.
- Enable the DAC channel using HAL_DAC_Start() or HAL_DAC_Start_DMA functions

Polling mode IO operation

- Start the DAC peripheral using HAL_DAC_Start()
- To read the DAC last data output value, use the HAL_DAC_GetValue() function.
- Stop the DAC peripheral using HAL_DAC_Stop()

DMA mode IO operation

- Start the DAC peripheral using HAL_DAC_Start_DMA(), at this stage the user specify the length of data to be transferred at each end of conversion

- At the middle of data transfer HAL_DACEx_ConvHalfCpltCallbackCh1() or HAL_DACEx_ConvHalfCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvHalfCpltCallbackCh1 or HAL_DAC_ConvHalfCpltCallbackCh2
- At The end of data transfer HAL_DAC_ConvCpltCallbackCh1() or HAL_DAC_ConvCpltCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ConvCpltCallbackCh1 or HAL_DAC_ConvCpltCallbackCh2
- In case of transfer Error, HAL_DAC_ErrorCallbackCh1() or HAL_DACEx_ErrorCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1 or HAL_DACEx_ErrorCallbackCh2
- For STM32F100x devices with specific feature: DMA underrun. In case of DMA underrun, DAC interruption triggers and execute internal function HAL_DAC_IRQHandler. HAL_DAC_DMAUnderrunCallbackCh1() or HAL_DACEx_DMAUnderrunCallbackCh2() function is executed and user can add his own code by customization of function pointer HAL_DAC_DMAUnderrunCallbackCh1 or HAL_DACEx_DMAUnderrunCallbackCh2 add his own code by customization of function pointer HAL_DAC_ErrorCallbackCh1
- Stop the DAC peripheral using HAL_DAC_Stop_DMA()

DAC HAL driver macros list

Below the list of most used macros in DAC HAL driver.

- __HAL_DAC_ENABLE : Enable the DAC peripheral
- __HAL_DAC_DISABLE : Disable the DAC peripheral
- __HAL_DAC_CLEAR_FLAG: Clear the DAC's pending flags
- __HAL_DAC_GET_FLAG: Get the selected DAC's flag status



You can refer to the DAC HAL driver header file for more useful macros

12.3.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize and configure the DAC.
- De-initialize the DAC.

This section contains the following APIs:

- [*HAL_DAC_Init\(\)*](#)
- [*HAL_DAC_DeInit\(\)*](#)
- [*HAL_DAC_MspInit\(\)*](#)
- [*HAL_DAC_MspDeInit\(\)*](#)

12.3.4 IO operation functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.

- Get result of conversion.

This section contains the following APIs:

- [HAL_DAC_Start\(\)](#)
- [HAL_DAC_Stop\(\)](#)
- [HAL_DAC_Start_DMA\(\)](#)
- [HAL_DAC_Stop_DMA\(\)](#)
- [HAL_DAC_GetValue\(\)](#)
- [HAL_DAC_IRQHandler\(\)](#)
- [HAL_DAC_ConvCpltCallbackCh1\(\)](#)
- [HAL_DAC_ConvHalfCpltCallbackCh1\(\)](#)
- [HAL_DAC_ErrorCallbackCh1\(\)](#)
- [HAL_DAC_DMAUnderrunCallbackCh1\(\)](#)
- [HAL_DAC_SetValue\(\)](#)
- [HAL_DAC_ConfigChannel\(\)](#)
- [HAL_DAC_GetState\(\)](#)
- [HAL_DAC_GetError\(\)](#)

12.3.5 Peripheral Control functions

This section provides functions allowing to:

- Configure channels.
- Set the specified data holding register value for DAC channel.

This section contains the following APIs:

- [HAL_DAC_ConfigChannel\(\)](#)
- [HAL_DAC_SetValue\(\)](#)

12.3.6 Peripheral State and Errors functions

This subsection provides functions allowing to

- Check the DAC state.
- Check the DAC Errors.

This section contains the following APIs:

- [HAL_DAC_GetState\(\)](#)
- [HAL_DAC_GetError\(\)](#)

12.3.7 HAL_DAC_Init

Function Name	HAL_StatusTypeDef HAL_DAC_Init (DAC_HandleTypeDef * hdac)
Function Description	Initializes the DAC peripheral according to the specified parameters in the DAC_InitStruct.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • HAL status

12.3.8 HAL_DAC_DeInit

Function Name	HAL_StatusTypeDef HAL_DAC_DeInit (DAC_HandleTypeDef * hdac)
---------------	--

Function Description	Deinitializes the DAC peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • HAL status

12.3.9 HAL_DAC_MspInit

Function Name	void HAL_DAC_MspInit (DAC_HandleTypeDef * hdac)
Function Description	Initializes the DAC MSP.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • None

12.3.10 HAL_DAC_MspDeInit

Function Name	void HAL_DAC_MspDeInit (DAC_HandleTypeDef * hdac)
Function Description	DeInitializes the DAC MSP.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • None

12.3.11 HAL_DAC_Start

Function Name	HAL_StatusTypeDef HAL_DAC_Start (DAC_HandleTypeDef * hdac, uint32_t Channel)
Function Description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> • HAL status

12.3.12 HAL_DAC_Stop

Function Name	HAL_StatusTypeDef HAL_DAC_Stop (DAC_HandleTypeDef * hdac, uint32_t Channel)
Function Description	Disables DAC and stop conversion of channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> • HAL status

12.3.13 HAL_DAC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_DAC_Start_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t * pData, uint32_t Length, uint32_t Alignment)
Function Description	Enables DAC and starts conversion of channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected • pData: The destination peripheral Buffer address. • Length: The length of data to be transferred from memory to DAC peripheral • Alignment: Specifies the data alignment for DAC channel. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selectedDAC_ALIGN_12B_L: 12bit left data alignment selectedDAC_ALIGN_12B_R: 12bit right data alignment selected
Return values	<ul style="list-style-type: none"> • HAL status

12.3.14 HAL_DAC_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_DAC_Stop_DMA (DAC_HandleTypeDef * hdac, uint32_t Channel)
Function Description	Disables DAC and stop conversion of channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> • HAL status

12.3.15 HAL_DAC_GetValue

Function Name	uint32_t HAL_DAC_GetValue (DAC_HandleTypeDef * hdac, uint32_t Channel)
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> • The selected DAC channel data output value.

12.3.16 HAL_DAC_IRQHandler

Function Name	void HAL_DAC_IRQHandler (DAC_HandleTypeDef * hdac)
Function Description	Handles DAC interrupt request.
Parameters	<ul style="list-style-type: none"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None

12.3.17 HAL_DAC_ConvCpltCallbackCh1

Function Name	void HAL_DAC_ConvCpltCallbackCh1 (DAC_HandleTypeDef * hdac)
Function Description	Conversion complete callback in non blocking mode for Channel1.
Parameters	<ul style="list-style-type: none"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None

12.3.18 HAL_DAC_ConvHalfCpltCallbackCh1

Function Name	void HAL_DAC_ConvHalfCpltCallbackCh1 (DAC_HandleTypeDef * hdac)
Function Description	Conversion half DMA transfer callback in non blocking mode for Channel1.
Parameters	<ul style="list-style-type: none"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None

12.3.19 HAL_DAC_ErrorCallbackCh1

Function Name	void HAL_DAC_ErrorCallbackCh1 (DAC_HandleTypeDef * hdac)
Function Description	Error DAC callback for Channel1.
Parameters	<ul style="list-style-type: none"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None

12.3.20 HAL_DAC_DMAUnderrunCallbackCh1

Function Name	void HAL_DAC_DMAUnderrunCallbackCh1 (DAC_HandleTypeDef * hdac)
Function Description	DMA underrun DAC callback for channel1.
Parameters	<ul style="list-style-type: none"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None

12.3.21 HAL_DAC_SetValue

Function Name	HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t
---------------	---

Alignment, uint32_t Data)

Function Description	Set the specified data holding register value for DAC channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected • Alignment: Specifies the data alignment. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selectedDAC_ALIGN_12B_L: 12bit left data alignment selectedDAC_ALIGN_12B_R: 12bit right data alignment selected • Data: Data to be loaded in the selected data holding register.
Return values	<ul style="list-style-type: none"> • HAL status

12.3.22 HAL_DAC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)
Function Description	Configures the selected DAC channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • sConfig: DAC configuration structure. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> • HAL status

12.3.23 HAL_DAC_GetState

Function Name	HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)
Function Description	return the DAC state
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • HAL state

12.3.24 HAL_DAC_GetError

Function Name	uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)
Function Description	Return the DAC error code.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • DAC Error Code

12.3.25 HAL_DAC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_DAC_ConfigChannel (DAC_HandleTypeDef * hdac, DAC_ChannelConfTypeDef * sConfig, uint32_t Channel)
Function Description	Configures the selected DAC channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • sConfig: DAC configuration structure. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected
Return values	<ul style="list-style-type: none"> • HAL status

12.3.26 HAL_DAC_SetValue

Function Name	HAL_StatusTypeDef HAL_DAC_SetValue (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)
Function Description	Set the specified data holding register value for DAC channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1: DAC Channel1 selectedDAC_CHANNEL_2: DAC Channel2 selected • Alignment: Specifies the data alignment. This parameter can be one of the following values: DAC_ALIGN_8B_R: 8bit right data alignment selectedDAC_ALIGN_12B_L: 12bit left data alignment selectedDAC_ALIGN_12B_R: 12bit right data alignment selected • Data: Data to be loaded in the selected data holding register.
Return values	<ul style="list-style-type: none"> • HAL status

12.3.27 HAL_DAC_GetState

Function Name	HAL_DAC_StateTypeDef HAL_DAC_GetState (DAC_HandleTypeDef * hdac)
Function Description	return the DAC state
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • HAL state

12.3.28 HAL_DAC_GetError

Function Name	uint32_t HAL_DAC_GetError (DAC_HandleTypeDef * hdac)
Function Description	Return the DAC error code.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that

contains the configuration information for the specified DAC.

Return values

- DAC Error Code

12.4 DAC Firmware driver defines

12.4.1 DAC

DAC Channel selection

DAC_CHANNEL_1

DAC_CHANNEL_2

DAC data alignment

DAC_ALIGN_12B_R

DAC_ALIGN_12B_L

DAC_ALIGN_8B_R

DAC Error Code

HAL_DAC_ERROR_NONE

No error

HAL_DAC_ERROR_DMAUNDERRUNCH1

DAC channel1 DMA underrun error

HAL_DAC_ERROR_DMAUNDERRUNCH2

DAC channel2 DMA underrun error

HAL_DAC_ERROR_DMA

DMA error

DAC Exported Macros

__HAL_DAC_RESET_HANDLE_STATE

Description:

- Reset DAC handle state.

Parameters:

- __HANDLE__: specifies the DAC handle.

Return value:

- None

__HAL_DAC_ENABLE

Description:

- Enable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle.
- __DAC_Channel__: specifies the DAC channel

Return value:

- None

__HAL_DAC_DISABLE

Description:

- Disable the DAC channel.

Parameters:

- __HANDLE__: specifies the DAC handle
- __DAC_Channel__: specifies the DAC

channel.

Return value:

- None

Description:

- Enable the DAC interrupt.

Parameters:

- `__HANDLE__`: specifies the DAC handle
- `__INTERRUPT__`: specifies the DAC interrupt. This parameter can be any combination of the following values:
 - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt
 - `DAC_IT_DMAUDR2`: DAC channel 2 DMA underrun interrupt

Return value:

- None

Description:

- Disable the DAC interrupt.

Parameters:

- `__HANDLE__`: specifies the DAC handle
- `__INTERRUPT__`: specifies the DAC interrupt. This parameter can be any combination of the following values:
 - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt
 - `DAC_IT_DMAUDR2`: DAC channel 2 DMA underrun interrupt

Return value:

- None

Description:

- Checks if the specified DAC interrupt source is enabled or disabled.

Parameters:

- `__HANDLE__`: DAC handle
- `__INTERRUPT__`: DAC interrupt source to check This parameter can be any combination of the following values:
 - `DAC_IT_DMAUDR1`: DAC channel 1 DMA underrun interrupt
 - `DAC_IT_DMAUDR2`: DAC channel 2 DMA underrun interrupt

Return value:

- State: of interruption (SET or RESET)

`__HAL_DAC_ENABLE_IT`

`__HAL_DAC_DISABLE_IT`

`__HAL_DAC_GET_IT_SOURCE`

__HAL_DAC_GET_FLAG**Description:**

- Get the selected DAC's flag status.

Parameters:

- **__HANDLE__**: specifies the DAC handle.
- **__FLAG__**: specifies the DAC flag to get. This parameter can be any combination of the following values:
 - **DAC_FLAG_DMAUDR1**: DAC channel 1 DMA underrun flag
 - **DAC_FLAG_DMAUDR2**: DAC channel 2 DMA underrun flag

Return value:

- None

__HAL_DAC_CLEAR_FLAG**Description:**

- Clear the DAC's flag.

Parameters:

- **__HANDLE__**: specifies the DAC handle.
- **__FLAG__**: specifies the DAC flag to clear. This parameter can be any combination of the following values:
 - **DAC_FLAG_DMAUDR1**: DAC channel 1 DMA underrun flag
 - **DAC_FLAG_DMAUDR2**: DAC channel 2 DMA underrun flag

Return value:

- None

DAC flags definition

DAC_FLAG_DMAUDR1

DAC_FLAG_DMAUDR2

DAC IT definition

DAC_IT_DMAUDR1

DAC_IT_DMAUDR2

DAC output buffer

DAC_OUTPUTBUFFER_ENABLE

DAC_OUTPUTBUFFER_DISABLE

DAC Private Macros

IS_DAC_TRIGGER

IS_DAC_OUTPUT_BUFFER_STATE

IS_DAC_CHANNEL

IS_DAC_ALIGN

IS_DAC_DATA

DAC_DHR12R1_ALIGNMENT
DAC_DHR12R2_ALIGNMENT
DAC_DHR12RD_ALIGNMENT

DAC trigger selection

DAC_TRIGGER_NONE	Conversion is automatic once the DAC1_DHRxxxx register has been loaded, and not by external trigger
DAC_TRIGGER_T6_TRGO	TIM6 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T7_TRGO	TIM7 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T9_TRGO	TIM9 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T2_TRGO	TIM2 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_T4_TRGO	TIM4 TRGO selected as external conversion trigger for DAC channel
DAC_TRIGGER_EXT_IT9	EXTI Line9 event selected as external conversion trigger for DAC channel
DAC_TRIGGER_SOFTWARE	Conversion started by software trigger for DAC channel

13 HAL DAC Extension Driver

13.1 HAL DAC Extension Driver

13.2 DACEx Firmware driver API description

13.2.1 How to use this driver

- When Dual mode is enabled (i.e DAC Channel1 and Channel2 are used simultaneously) : Use HAL_DACEx_DualGetValue() to get digital data to be converted and use HAL_DACEx_DualSetValue() to set digital value to converted simultaneously in Channel 1 and Channel 2.
- Use HAL_DACEx_TriangleWaveGenerate() to generate Triangle signal.
- Use HAL_DACEx_NoiseWaveGenerate() to generate Noise signal.

13.2.2 Extended features functions

This section provides functions allowing to:

- Start conversion.
- Stop conversion.
- Start conversion and enable DMA transfer.
- Stop conversion and disable DMA transfer.
- Get result of conversion.
- Get result of dual mode conversion.

This section contains the following APIs:

- [HAL_DACEx_DualGetValue\(\)](#)
- [HAL_DACEx_TriangleWaveGenerate\(\)](#)
- [HAL_DACEx_NoiseWaveGenerate\(\)](#)
- [HAL_DACEx_DualSetValue\(\)](#)
- [HAL_DACEx_ConvCpltCallbackCh2\(\)](#)
- [HAL_DACEx_ConvHalfCpltCallbackCh2\(\)](#)
- [HAL_DACEx_ErrorCallbackCh2\(\)](#)
- [HAL_DACEx_DMAUnderrunCallbackCh2\(\)](#)

13.2.3 HAL_DACEx_DualGetValue

Function Name	uint32_t HAL_DACEx_DualGetValue (DAC_HandleTypeDef * hdac)
Function Description	Returns the last data output value of the selected DAC channel.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> • The selected DAC channel data output value.

13.2.4 HAL_DACEx_TriangleWaveGenerate

Function Name	HAL_StatusTypeDef HAL_DACEx_TriangleWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t
---------------	---

	Amplitude)
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2 • Amplitude: Select max triangle amplitude. This parameter can be one of the following values: DAC_TRIANGLEAMPLITUDE_1: Select max triangle amplitude of 1 DAC_TRIANGLEAMPLITUDE_3: Select max triangle amplitude of 3 DAC_TRIANGLEAMPLITUDE_7: Select max triangle amplitude of 7 DAC_TRIANGLEAMPLITUDE_15: Select max triangle amplitude of 15 DAC_TRIANGLEAMPLITUDE_31: Select max triangle amplitude of 31 DAC_TRIANGLEAMPLITUDE_63: Select max triangle amplitude of 63 DAC_TRIANGLEAMPLITUDE_127: Select max triangle amplitude of 127 DAC_TRIANGLEAMPLITUDE_255: Select max triangle amplitude of 255 DAC_TRIANGLEAMPLITUDE_511: Select max triangle amplitude of 511 DAC_TRIANGLEAMPLITUDE_1023: Select max triangle amplitude of 1023 DAC_TRIANGLEAMPLITUDE_2047: Select max triangle amplitude of 2047 DAC_TRIANGLEAMPLITUDE_4095: Select max triangle amplitude of 4095
Return values	<ul style="list-style-type: none"> • HAL status

13.2.5 HAL_DACEx_NoiseWaveGenerate

Function Name	HAL_StatusTypeDef HAL_DACEx_NoiseWaveGenerate (DAC_HandleTypeDef * hdac, uint32_t Channel, uint32_t Amplitude)
Function Description	Enables or disables the selected DAC channel wave generation.
Parameters	<ul style="list-style-type: none"> • hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC. • Channel: The selected DAC channel. This parameter can be one of the following values: DAC_CHANNEL_1 / DAC_CHANNEL_2 • Amplitude: Unmask DAC channel LFSR for noise wave generation. This parameter can be one of the following values: DAC_LFSRUNMASK_BIT0: Unmask DAC channel LFSR bit0 for noise wave generation DAC_LFSRUNMASK_BITS1_0: Unmask DAC channel LFSR bit[1:0] for noise wave generation DAC_LFSRUNMASK_BITS2_0: Unmask DAC channel LFSR bit[2:0] for noise wave generation DAC_LFSRUNMASK_BITS3_0: Unmask DAC channel LFSR bit[3:0] for noise wave generation DAC_LFSRUNMASK_BITS4_0: Unmask DAC channel LFSR bit[4:0] for noise wave generation DAC_LFSRUNMASK_BITS5_0: Unmask DAC

channel LFSR bit[5:0] for noise wave generation
 generationDAC_LFSRUNMASK_BITS6_0: Unmask DAC channel LFSR bit[6:0] for noise wave generation
 generationDAC_LFSRUNMASK_BITS7_0: Unmask DAC channel LFSR bit[7:0] for noise wave generation
 generationDAC_LFSRUNMASK_BITS8_0: Unmask DAC channel LFSR bit[8:0] for noise wave generation
 generationDAC_LFSRUNMASK_BITS9_0: Unmask DAC channel LFSR bit[9:0] for noise wave generation
 generationDAC_LFSRUNMASK_BITS10_0: Unmask DAC channel LFSR bit[10:0] for noise wave generation
 generationDAC_LFSRUNMASK_BITS11_0: Unmask DAC channel LFSR bit[11:0] for noise wave generation

Return values

- HAL status

13.2.6 HAL_DACEx_DualSetValue

Function Name `HAL_StatusTypeDef HAL_DACEx_DualSetValue (DAC_HandleTypeDef * hdac, uint32_t Alignment, uint32_t Data1, uint32_t Data2)`

Function Description Set the specified data holding register value for dual DAC channel.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
- **Alignment:** Specifies the data alignment for dual channel DAC. This parameter can be one of the following values:
 DAC_ALIGN_8B_R: 8bit right data alignment selected
 DAC_ALIGN_12B_L: 12bit left data alignment selected
 DAC_ALIGN_12B_R: 12bit right data alignment selected
- **Data1:** Data for DAC Channel2 to be loaded in the selected data holding register.
- **Data2:** Data for DAC Channel1 to be loaded in the selected data holding register.

Return values

- HAL status

Notes

- In dual mode, a unique register access is required to write in both DAC channels at the same time.

13.2.7 HAL_DACEx_ConvCpltCallbackCh2

Function Name `void HAL_DACEx_ConvCpltCallbackCh2 (DAC_HandleTypeDef * hdac)`

Function Description Conversion complete callback in non blocking mode for Channel2.

Parameters

- **hdac:** pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.

Return values

- None

13.2.8 HAL_DACEx_ConvHalfCpltCallbackCh2

Function Name `void HAL_DACEx_ConvHalfCpltCallbackCh2 (DAC_HandleTypeDef * hdac)`

Function Description	Conversion half DMA transfer callback in non blocking mode for Channel2.
Parameters	<ul style="list-style-type: none"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None

13.2.9 HAL_DACEx_ErrorCallbackCh2

Function Name	void HAL_DACEx_ErrorCallbackCh2 (DAC_HandleTypeDef *hdac)
Function Description	Error DAC callback for Channel2.
Parameters	<ul style="list-style-type: none"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None

13.2.10 HAL_DACEx_DMAUnderrunCallbackCh2

Function Name	void HAL_DACEx_DMAUnderrunCallbackCh2 (DAC_HandleTypeDef *hdac)
Function Description	DMA underrun DAC callback for channel2.
Parameters	<ul style="list-style-type: none"> hdac: pointer to a DAC_HandleTypeDef structure that contains the configuration information for the specified DAC.
Return values	<ul style="list-style-type: none"> None

13.3 DACEx Firmware driver defines

13.3.1 DACEx

DACEx Ifsrunmask triangleamplitude

DAC_LFSRUNMASK_BIT0	Unmask DAC channel LFSR bit0 for noise wave generation
DAC_LFSRUNMASK_BITS1_0	Unmask DAC channel LFSR bit[1:0] for noise wave generation
DAC_LFSRUNMASK_BITS2_0	Unmask DAC channel LFSR bit[2:0] for noise wave generation
DAC_LFSRUNMASK_BITS3_0	Unmask DAC channel LFSR bit[3:0] for noise wave generation
DAC_LFSRUNMASK_BITS4_0	Unmask DAC channel LFSR bit[4:0] for noise wave generation
DAC_LFSRUNMASK_BITS5_0	Unmask DAC channel LFSR bit[5:0] for noise wave generation
DAC_LFSRUNMASK_BITS6_0	Unmask DAC channel LFSR bit[6:0] for noise wave generation
DAC_LFSRUNMASK_BITS7_0	Unmask DAC channel LFSR bit[7:0] for noise wave generation

DAC_LFSRUNMASK_BITS8_0	Unmask DAC channel LFSR bit[8:0] for noise wave generation
DAC_LFSRUNMASK_BITS9_0	Unmask DAC channel LFSR bit[9:0] for noise wave generation
DAC_LFSRUNMASK_BITS10_0	Unmask DAC channel LFSR bit[10:0] for noise wave generation
DAC_LFSRUNMASK_BITS11_0	Unmask DAC channel LFSR bit[11:0] for noise wave generation
DAC_TRIANGLEAMPLITUDE_1	Select max triangle amplitude of 1
DAC_TRIANGLEAMPLITUDE_3	Select max triangle amplitude of 3
DAC_TRIANGLEAMPLITUDE_7	Select max triangle amplitude of 7
DAC_TRIANGLEAMPLITUDE_15	Select max triangle amplitude of 15
DAC_TRIANGLEAMPLITUDE_31	Select max triangle amplitude of 31
DAC_TRIANGLEAMPLITUDE_63	Select max triangle amplitude of 63
DAC_TRIANGLEAMPLITUDE_127	Select max triangle amplitude of 127
DAC_TRIANGLEAMPLITUDE_255	Select max triangle amplitude of 255
DAC_TRIANGLEAMPLITUDE_511	Select max triangle amplitude of 511
DAC_TRIANGLEAMPLITUDE_1023	Select max triangle amplitude of 1023
DAC_TRIANGLEAMPLITUDE_2047	Select max triangle amplitude of 2047
DAC_TRIANGLEAMPLITUDE_4095	Select max triangle amplitude of 4095
IS_DAC_LFSR_UNMASK_TRIANGLE_AMPLITUDE	
<i>DACEx wave generation</i>	
DAC_WAVE_NOISE	
DAC_WAVE_TRIANGLE	

14 HAL DMA Generic Driver

14.1 HAL DMA Generic Driver

14.2 DMA Firmware driver registers structures

14.2.1 DMA_InitTypeDef

Data Fields

- *uint32_t Direction*
- *uint32_t PeriphInc*
- *uint32_t MemInc*
- *uint32_t PeriphDataAlignment*
- *uint32_t MemDataAlignment*
- *uint32_t Mode*
- *uint32_t Priority*

Field Documentation

- *uint32_t DMA_InitTypeDef::Direction*
Specifies if the data will be transferred from memory to peripheral, from memory to memory or from peripheral to memory. This parameter can be a value of [DMA_Data_transfer_direction](#)
- *uint32_t DMA_InitTypeDef::PeriphInc*
Specifies whether the Peripheral address register should be incremented or not. This parameter can be a value of [DMA_Peripheral_incremented_mode](#)
- *uint32_t DMA_InitTypeDef::MemInc*
Specifies whether the memory address register should be incremented or not. This parameter can be a value of [DMA_Memory_incremented_mode](#)
- *uint32_t DMA_InitTypeDef::PeriphDataAlignment*
Specifies the Peripheral data width. This parameter can be a value of [DMA_Peripheral_data_size](#)
- *uint32_t DMA_InitTypeDef::MemDataAlignment*
Specifies the Memory data width. This parameter can be a value of [DMA_Memory_data_size](#)
- *uint32_t DMA_InitTypeDef::Mode*
Specifies the operation mode of the DMAy Channelx. This parameter can be a value of [DMA_mode](#)
Note: The circular buffer mode cannot be used if the memory-to-memory data transfer is configured on the selected Channel
- *uint32_t DMA_InitTypeDef::Priority*
Specifies the software priority for the DMAy Channelx. This parameter can be a value of [DMA_Priority_level](#)

14.2.2 __DMA_HandleTypeDef

Data Fields

- ***DMA_Channel_TypeDef * Instance***
- ***DMA_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***HAL_DMA_StateTypeDef State***
- ***void * Parent***
- ***void(* XferCpltCallback***
- ***void(* XferHalfCpltCallback***
- ***void(* XferErrorCallback***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***DMA_Channel_TypeDef* __DMA_HandleTypeDef::Instance***
Register base address
- ***DMA_InitTypeDef __DMA_HandleTypeDef::Init***
DMA communication parameters
- ***HAL_LockTypeDef __DMA_HandleTypeDef::Lock***
DMA locking object
- ***HAL_DMA_StateTypeDef __DMA_HandleTypeDef::State***
DMA transfer state
- ***void* __DMA_HandleTypeDef::Parent***
Parent object state
- ***void(* __DMA_HandleTypeDef::XferCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
DMA transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferHalfCpltCallback)(struct __DMA_HandleTypeDef *hdma)***
DMA Half transfer complete callback
- ***void(* __DMA_HandleTypeDef::XferErrorCallback)(struct __DMA_HandleTypeDef *hdma)***
DMA transfer error callback
- ***__IO uint32_t __DMA_HandleTypeDef::ErrorCode***
DMA Error code

14.3 DMA Firmware driver API description

14.3.1 How to use this driver

1. Enable and configure the peripheral to be connected to the DMA Channel (except for internal SRAM / FLASH memories: no initialization is necessary) please refer to Reference manual for connection between peripherals and DMA requests.
2. For a given Channel, program the required configuration through the following parameters: Transfer Direction, Source and Destination data formats, Circular or Normal mode, Channel Priority level, Source and Destination Increment mode, using HAL_DMA_Init() function.
3. Use HAL_DMA_GetState() function to return the DMA state and HAL_DMA_GetError() in case of error detection.
4. Use HAL_DMA_Abort() function to abort the current transfer In Memory-to-Memory transfer mode, Circular mode is not allowed.

Polling mode IO operation

- Use `HAL_DMA_Start()` to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred
- Use `HAL_DMA_PollForTransfer()` to poll for the end of current transfer, in this case a fixed Timeout can be configured by User depending from his application.

Interrupt mode IO operation

- Configure the DMA interrupt priority using `HAL_NVIC_SetPriority()`
- Enable the DMA IRQ handler using `HAL_NVIC_EnableIRQ()`
- Use `HAL_DMA_Start_IT()` to start DMA transfer after the configuration of Source address and destination address and the Length of data to be transferred. In this case the DMA interrupt is configured
- Use `HAL_DMAy_Channelx_IRQHandler()` called under `DMA_IRQHandler()` Interrupt subroutine
- At the end of data transfer `HAL_DMA_IRQHandler()` function is executed and user can add his own function by customization of function pointer `XferCpltCallback` and `XferErrorCallback` (i.e a member of DMA handle structure).

DMA HAL driver macros list

Below the list of most used macros in DMA HAL driver.

- `__HAL_DMA_ENABLE`: Enable the specified DMA Channel.
- `__HAL_DMA_DISABLE`: Disable the specified DMA Channel.
- `__HAL_DMA_GET_FLAG`: Get the DMA Channel pending flags.
- `__HAL_DMA_CLEAR_FLAG`: Clear the DMA Channel pending flags.
- `__HAL_DMA_ENABLE_IT`: Enable the specified DMA Channel interrupts.
- `__HAL_DMA_DISABLE_IT`: Disable the specified DMA Channel interrupts.
- `__HAL_DMA_GET_IT_SOURCE`: Check whether the specified DMA Channel interrupt has occurred or not.



You can refer to the DMA HAL driver header file for more useful macros

14.3.2 Initialization and de-initialization functions

This section provides functions allowing to initialize the DMA Channel source and destination addresses, incrementation and data sizes, transfer direction, circular/normal mode selection, memory-to-memory mode selection and Channel priority value.

The `HAL_DMA_Init()` function follows the DMA configuration procedures as described in reference manual.

This section contains the following APIs:

- [*HAL_DMA_Init\(\)*](#)
- [*HAL_DMA_DeInit\(\)*](#)

14.3.3 IO operation functions

This section provides functions allowing to:

- Configure the source, destination address and data length and Start DMA transfer
- Configure the source, destination address and data length and Start DMA transfer with interrupt
- Abort DMA transfer
- Poll for transfer complete
- Handle DMA interrupt request

This section contains the following APIs:

- [HAL_DMA_Start\(\)](#)
- [HAL_DMA_Start_IT\(\)](#)
- [HAL_DMA_Abort\(\)](#)
- [HAL_DMA_PollForTransfer\(\)](#)
- [HAL_DMA_IRQHandler\(\)](#)

14.3.4 State and Errors functions

This subsection provides functions allowing to

- Check the DMA state
- Get error code

This section contains the following APIs:

- [HAL_DMA_GetState\(\)](#)
- [HAL_DMA_GetError\(\)](#)

14.3.5 HAL_DMA_Init

Function Name	HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef * hdma)
Function Description	Initializes the DMA according to the specified parameters in the DMA_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hdma: Pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> • HAL status

14.3.6 HAL_DMA_DeInit

Function Name	HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef * hdma)
Function Description	DeInitializes the DMA peripheral.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> • HAL status

14.3.7 HAL_DMA_Start

Function Name	HAL_StatusTypeDef HAL_DMA_Start (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)
---------------	--

Function Description	Starts the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hdma: : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • SrcAddress: The source memory Buffer address • DstAddress: The destination memory Buffer address • DataLength: The length of data to be transferred from source to destination
Return values	<ul style="list-style-type: none"> • HAL status

14.3.8 HAL_DMA_Start_IT

Function Name	HAL_StatusTypeDef HAL_DMA_Start_IT (DMA_HandleTypeDef * hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)
Function Description	Start the DMA Transfer with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel. • SrcAddress: The source memory Buffer address • DstAddress: The destination memory Buffer address • DataLength: The length of data to be transferred from source to destination
Return values	<ul style="list-style-type: none"> • HAL status

14.3.9 HAL_DMA_Abort

Function Name	HAL_StatusTypeDef HAL_DMA_Abort (DMA_HandleTypeDef * hdma)
Function Description	Aborts the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hdma: : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • After disabling a DMA Channel, a check for wait until the DMA Channel is effectively disabled is added. If a Channel is disabled while a data transfer is ongoing, the current data will be transferred and the Channel will be effectively disabled only after the transfer of this single data is finished.

14.3.10 HAL_DMA_PollForTransfer

Function Name	HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef * hdma, uint32_t CompleteLevel, uint32_t Timeout)
Function Description	Polling for transfer complete.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.

- **CompleteLevel:** Specifies the DMA level complete.
 - **Timeout:** Timeout duration.
- Return values
- HAL status

14.3.11 HAL_DMA_IRQHandler

Function Name	void HAL_DMA_IRQHandler (DMA_HandleTypeDef * hdma)
Function Description	Handles DMA interrupt request.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> • None

14.3.12 HAL_DMA_GetState

Function Name	HAL_DMA_StateTypeDef HAL_DMA_GetState (DMA_HandleTypeDef * hdma)
Function Description	Returns the DMA state.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> • HAL state

14.3.13 HAL_DMA_GetError

Function Name	uint32_t HAL_DMA_GetError (DMA_HandleTypeDef * hdma)
Function Description	Return the DMA error code.
Parameters	<ul style="list-style-type: none"> • hdma: : pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA Channel.
Return values	<ul style="list-style-type: none"> • DMA Error Code

14.4 DMA Firmware driver defines

14.4.1 DMA

DMA Data transfer direction

DMA_PERIPH_TO_MEMORY	Peripheral to memory direction
DMA_MEMORY_TO_PERIPH	Memory to peripheral direction
DMA_MEMORY_TO_MEMORY	Memory to memory direction

DMA Error Code

HAL_DMA_ERROR_NONE	No error
HAL_DMA_ERROR_TE	Transfer error
HAL_DMA_ERROR_TIMEOUT	Timeout error

DMA Exported Macros

<code>__HAL_DMA_RESET_HANDLE_STATE</code>	<p>Description:</p> <ul style="list-style-type: none">Reset DMA handle state. <p>Parameters:</p> <ul style="list-style-type: none"><code>__HANDLE__</code>: DMA handle. <p>Return value:</p> <ul style="list-style-type: none">None
<code>__HAL_DMA_ENABLE</code>	<p>Description:</p> <ul style="list-style-type: none">Enable the specified DMA Channel. <p>Parameters:</p> <ul style="list-style-type: none"><code>__HANDLE__</code>: DMA handle <p>Return value:</p> <ul style="list-style-type: none">None.
<code>__HAL_DMA_DISABLE</code>	<p>Description:</p> <ul style="list-style-type: none">Disable the specified DMA Channel. <p>Parameters:</p> <ul style="list-style-type: none"><code>__HANDLE__</code>: DMA handle <p>Return value:</p> <ul style="list-style-type: none">None.
<code>__HAL_DMA_ENABLE_IT</code>	<p>Description:</p> <ul style="list-style-type: none">Enables the specified DMA Channel interrupts. <p>Parameters:</p> <ul style="list-style-type: none"><code>__HANDLE__</code>: DMA handle<code>__INTERRUPT__</code>: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:<ul style="list-style-type: none">DMA_IT_TC: Transfer complete interrupt maskDMA_IT_HT: Half transfer complete interrupt maskDMA_IT_TE: Transfer error interrupt mask <p>Return value:</p> <ul style="list-style-type: none">None
<code>__HAL_DMA_DISABLE_IT</code>	<p>Description:</p> <ul style="list-style-type: none">Disables the specified DMA Channel interrupts. <p>Parameters:</p> <ul style="list-style-type: none"><code>__HANDLE__</code>: DMA handle

__HAL_DMA_GET_IT_SOURCE

DMA flag definitions

DMA_FLAG_GL1
DMA_FLAG_TC1
DMA_FLAG_HT1
DMA_FLAG_TE1
DMA_FLAG_GL2
DMA_FLAG_TC2
DMA_FLAG_HT2
DMA_FLAG_TE2
DMA_FLAG_GL3
DMA_FLAG_TC3
DMA_FLAG_HT3
DMA_FLAG_TE3
DMA_FLAG_GL4

- **__INTERRUPT__**: specifies the DMA interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
 - DMA_IT_TC: Transfer complete interrupt mask
 - DMA_IT_HT: Half transfer complete interrupt mask
 - DMA_IT_TE: Transfer error interrupt mask

Return value:

- None

Description:

- Checks whether the specified DMA Channel interrupt is enabled or disabled.

Parameters:

- **__HANDLE__**: DMA handle
- **__INTERRUPT__**: specifies the DMA interrupt source to check. This parameter can be one of the following values:
 - DMA_IT_TC: Transfer complete interrupt mask
 - DMA_IT_HT: Half transfer complete interrupt mask
 - DMA_IT_TE: Transfer error interrupt mask

Return value:

- The state of DMA_IT (SET or RESET).

DMA_FLAG_TC4

DMA_FLAG_HT4

DMA_FLAG_TE4

DMA_FLAG_GL5

DMA_FLAG_TC5

DMA_FLAG_HT5

DMA_FLAG_TE5

DMA_FLAG_GL6

DMA_FLAG_TC6

DMA_FLAG_HT6

DMA_FLAG_TE6

DMA_FLAG_GL7

DMA_FLAG_TC7

DMA_FLAG_HT7

DMA_FLAG_TE7

DMA interrupt enable definitions

DMA_IT_TC

DMA_IT_HT

DMA_IT_TE

DMA Memory data size

DMA_MDATAALIGN_BYTE Memory data alignment: Byte

DMA_MDATAALIGN_HALFWORD Memory data alignment: HalfWord

DMA_MDATAALIGN_WORD Memory data alignment: Word

DMA Memory incremented mode

DMA_MINC_ENABLE Memory increment mode Enable

DMA_MINC_DISABLE Memory increment mode Disable

DMA mode

DMA_NORMAL Normal mode

DMA_CIRCULAR Circular mode

DMA Peripheral data size

DMA_PDATAALIGN_BYTE Peripheral data alignment: Byte

DMA_PDATAALIGN_HALFWORD Peripheral data alignment: HalfWord

DMA_PDATAALIGN_WORD Peripheral data alignment: Word

DMA Peripheral incremented mode

DMA_PINC_ENABLE Peripheral increment mode Enable

DMA_PINC_DISABLE Peripheral increment mode Disable

DMA Priority level

DMA_PRIORITY_LOW	Priority level : Low
DMA_PRIORITY_MEDIUM	Priority level : Medium
DMA_PRIORITY_HIGH	Priority level : High
DMA_PRIORITY_VERY_HIGH	Priority level : Very_High

DMA Private Constants

HAL_TIMEOUT_DMA_ABORT

DMA Private Macros

IS_DMA_BUFFER_SIZE

IS_DMA_DIRECTION

IS_DMA_PERIPHERAL_INC_STATE

IS_DMA_MEMORY_INC_STATE

IS_DMA_PERIPHERAL_DATA_SIZE

IS_DMA_MEMORY_DATA_SIZE

IS_DMA_MODE

IS_DMA_PRIORITY

15 HAL DMA Extension Driver

15.1 HAL DMA Extension Driver

15.2 DMAEx Firmware driver defines

15.2.1 DMAEx

DMA Extended Exported Macros

<code>__HAL_DMA_GET_TC_FLAG_INDEX</code>	Description: <ul style="list-style-type: none"> Returns the current DMA Channel transfer complete flag. Parameters: <ul style="list-style-type: none"> <code>__HANDLE__</code>: DMA handle Return value: <ul style="list-style-type: none"> The: specified transfer complete flag index.
<code>__HAL_DMA_GET_HT_FLAG_INDEX</code>	Description: <ul style="list-style-type: none"> Returns the current DMA Channel half transfer complete flag. Parameters: <ul style="list-style-type: none"> <code>__HANDLE__</code>: DMA handle Return value: <ul style="list-style-type: none"> The: specified half transfer complete flag index.
<code>__HAL_DMA_GET_TE_FLAG_INDEX</code>	Description: <ul style="list-style-type: none"> Returns the current DMA Channel transfer error flag. Parameters: <ul style="list-style-type: none"> <code>__HANDLE__</code>: DMA handle Return value: <ul style="list-style-type: none"> The: specified transfer error flag index.
<code>__HAL_DMA_GET_FLAG</code>	Description: <ul style="list-style-type: none"> Get the DMA Channel pending flags. Parameters: <ul style="list-style-type: none"> <code>__HANDLE__</code>: DMA handle <code>__FLAG__</code>: Get the specified flag. This parameter can be any combination of the following values: <ul style="list-style-type: none"> <code>DMA_FLAG_TCx</code>: Transfer complete flag

- DMA_FLAG_HTx: Half transfer complete flag
- DMA_FLAG_TEx: Transfer error flag
Where x can be 1_7 or 1_5 to select the DMA Channel flag.

Return value:

- The: state of FLAG (SET or RESET).

Description:

- Clears the DMA Channel pending flags.

Parameters:

- __HANDLE__: DMA handle
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
 - DMA_FLAG_TCx: Transfer complete flag
 - DMA_FLAG_HTx: Half transfer complete flag
 - DMA_FLAG_TEx: Transfer error flag
Where x can be 1_7 or 1_5 to select the DMA Channel flag.

Return value:

- None

__HAL_DMA_CLEAR_FLAG

16 HAL FLASH Generic Driver

16.1 HAL FLASH Generic Driver

16.2 FLASH Firmware driver registers structures

16.2.1 FLASH_ProcessTypeDef

Data Fields

- *__IO FLASH_ProcedureTypeDef ProcedureOnGoing*
- *__IO uint32_t NbPagesToErase*
- *__IO uint32_t Page*
- *__IO uint32_t Address*
- *HAL_LockTypeDef Lock*
- *__IO uint32_t ErrorCode*

Field Documentation

- *__IO FLASH_ProcedureTypeDef FLASH_ProcessTypeDef::ProcedureOnGoing*
- *__IO uint32_t FLASH_ProcessTypeDef::NbPagesToErase*
- *__IO uint32_t FLASH_ProcessTypeDef::Page*
- *__IO uint32_t FLASH_ProcessTypeDef::Address*
- *HAL_LockTypeDef FLASH_ProcessTypeDef::Lock*
- *__IO uint32_t FLASH_ProcessTypeDef::ErrorCode*

16.3 FLASH Firmware driver API description

16.3.1 FLASH peripheral features

The Flash memory interface manages CPU AHB I-Code and D-Code accesses to the Flash memory. It implements the erase and program Flash memory operations and the read and write protection mechanisms.

The Flash memory interface accelerates code execution with a system of instruction prefetch.

The FLASH main features are:

- Flash memory read operations
- Flash memory program/erase operations
- Read / write protections
- Prefetch on I-Code
- Option Bytes programming

16.3.2 How to use this driver

This driver provides functions to configure and program the Flash memory of all STM32L1xx devices.

1. FLASH Memory Programming functions: this group includes all needed functions to erase and program the main memory:
 - Lock and Unlock the Flash interface.
 - Erase function: Erase Page.
 - Program functions: Fast Word and Half Page(should be executed from internal SRAM).
2. DATA EEPROM Programming functions: this group includes all needed functions to erase and program the DATA EEPROM memory:
 - Lock and Unlock the DATA EEPROM interface.
 - Erase function: Erase Byte, erase HalfWord, erase Word, erase Double Word (should be executed from internal SRAM).
 - Program functions: Fast Program Byte, Fast Program Half-Word, FastProgramWord, Program Byte, Program Half-Word, Program Word and Program Double-Word (should be executed from internal SRAM).
3. FLASH Option Bytes Programming functions: this group includes all needed functions to:
 - Lock and Unlock the Flash Option bytes.
 - Set/Reset the write protection.
 - Set the Read protection Level.
 - Set the BOR level.
 - Program the user option Bytes.
 - Launch the Option Bytes loader.
 - Get the Write protection.
 - Get the read protection status.
 - Get the BOR level.
 - Get the user option bytes.
4. Interrupts and flags management functions :
 - Handle FLASH interrupts by calling HAL_FLASH_IRQHandler()
 - Wait for last FLASH operation according to its status
 - Get error flag status by calling HAL_GetErrorCode()
5. FLASH Interface configuration functions: this group includes the management of following features:
 - Enable/Disable the RUN PowerDown mode.
 - Enable/Disable the SLEEP PowerDown mode.
6. FLASH Peripheral State methods: this group includes the management of following features:
 - Wait for the FLASH operation
 - Get the specific FLASH error flag

In addition to these function, this driver includes a set of macros allowing to handle the following operations:

- Set/Get the latency
- Enable/Disable the prefetch buffer
- Enable/Disable the 64 bit Read Access.
- Enable/Disable the Flash power-down
- Enable/Disable the FLASH interrupts
- Monitor the FLASH flags status

16.3.3 Programming operation functions

This subsection provides a set of functions allowing to manage the FLASH program operations.

The FLASH Memory Programming functions, includes the following functions:

- HAL_FLASH_Unlock(void);
- HAL_FLASH_Lock(void);
- HAL_FLASH_Program(uint32_t TypeProgram, uint32_t Address, uint32_t Data)
- HAL_FLASH_Program_IT(uint32_t TypeProgram, uint32_t Address, uint32_t Data)

Any operation of erase or program should follow these steps:

1. Call the HAL_FLASH_Unlock() function to enable the flash control register and program memory access.
2. Call the desired function to erase page or program data.
3. Call the HAL_FLASH_Lock() to disable the flash program memory access (recommended to protect the FLASH memory against possible unwanted operation).

16.3.4 Option Bytes Programming functions

The FLASH_Option Bytes Programming_functions, includes the following functions:

- HAL_FLASH_OB_Unlock(void);
- HAL_FLASH_OB_Lock(void);
- HAL_FLASH_OB_Launch(void);
- HAL_FLASHEx_OBProgram(FLASH_OBProgramInitTypeDef *pOBInit);
- HAL_FLASHEx_OBGetConfig(FLASH_OBProgramInitTypeDef *pOBInit);

Any operation of erase or program should follow these steps:

1. Call the HAL_FLASH_OB_Unlock() function to enable the Flash option control register access.
2. Call the following functions to program the desired option bytes.
 - HAL_FLASHEx_OBProgram(FLASH_OBProgramInitTypeDef *pOBInit);
3. Once all needed option bytes to be programmed are correctly written, call the HAL_FLASH_OB_Launch(void) function to launch the Option Bytes programming process.
4. Call the HAL_FLASH_OB_Lock() to disable the Flash option control register access (recommended to protect the option Bytes against possible unwanted operations).

Proprietary code Read Out Protection (PcROP):

1. The PcROP sector is selected by using the same option bytes as the Write protection. As a result, these 2 options are exclusive each other.
2. To activate PCROP mode for Flash sectors(s), you need to follow the sequence below:
 - Use this function HAL_FLASHEx_AdvOBProgram with PCROPState = OB_PCROP_STATE_ENABLE. *

16.3.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the FLASH memory operations.

This section contains the following APIs:

- [HAL_FLASH_Unlock\(\)](#)
- [HAL_FLASH_Lock\(\)](#)
- [HAL_FLASH_OB_Unlock\(\)](#)
- [HAL_FLASH_OB_Lock\(\)](#)
- [HAL_FLASH_OB_Launch\(\)](#)

16.3.6 Peripheral Errors functions

This subsection permit to get in run-time Errors of the FLASH peripheral.

This section contains the following APIs:

- [HAL_FLASH_GetError\(\)](#)

16.3.7 HAL_FLASH_Program

Function Name	HAL_StatusTypeDef HAL_FLASH_Program (uint32_t TypeProgram, uint32_t Address, uint64_t Data)
Function Description	Program word at a specified address.
Parameters	<ul style="list-style-type: none"> • TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program • Address: Specifies the address to be programmed. • Data: Specifies the data to be programmed
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> • To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation).

16.3.8 HAL_FLASH_Program_IT

Function Name	HAL_StatusTypeDef HAL_FLASH_Program_IT (uint32_t TypeProgram, uint32_t Address, uint64_t Data)
Function Description	Program word at a specified address with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • TypeProgram: Indicate the way to program at a specified address. This parameter can be a value of FLASH Type Program • Address: specifies the address to be programmed. • Data: specifies the data to be programmed
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status

16.3.9 HAL_FLASH_EndOfOperationCallback

Function Name	void HAL_FLASH_EndOfOperationCallback (uint32_t ReturnValue)
Function Description	FLASH end of operation interrupt callback.
Parameters	<ul style="list-style-type: none"> • ReturnValue: The value saved in this parameter depends on the ongoing procedure Pages Erase: Address of the page which has been erased (if 0xFFFFFFFF, it means that all the selected sectors have been erased) Program: Address which was selected for data program
Return values	<ul style="list-style-type: none"> • none

16.3.10 HAL_FLASH_OperationErrorCallback

Function Name	void HAL_FLASH_OperationErrorCallback (uint32_t ReturnValue)
Function Description	FLASH operation error interrupt callback.

Parameters	<ul style="list-style-type: none"> • ReturnValue: The value saved in this parameter depends on the ongoing procedurePages Erase: Address of the page which returned an errorProgram: Address which was selected for data program
Return values	<ul style="list-style-type: none"> • none

16.3.11 HAL_FLASH_IRQHandler

Function Name	void HAL_FLASH_IRQHandler (void)
Function Description	This function handles FLASH interrupt request.
Return values	<ul style="list-style-type: none"> • None

16.3.12 HAL_FLASH_Unlock

Function Name	HAL_StatusTypeDef HAL_FLASH_Unlock (void)
Function Description	Unlock the FLASH control register access.
Return values	<ul style="list-style-type: none"> • HAL Status

16.3.13 HAL_FLASH_Lock

Function Name	HAL_StatusTypeDef HAL_FLASH_Lock (void)
Function Description	Locks the FLASH control register access.
Return values	<ul style="list-style-type: none"> • HAL Status

16.3.14 HAL_FLASH_OB_Unlock

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Unlock (void)
Function Description	Unlock the FLASH Option Control Registers access.
Return values	<ul style="list-style-type: none"> • HAL Status

16.3.15 HAL_FLASH_OB_Lock

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Lock (void)
Function Description	Lock the FLASH Option Control Registers access.
Return values	<ul style="list-style-type: none"> • HAL Status

16.3.16 HAL_FLASH_OB_Launch

Function Name	HAL_StatusTypeDef HAL_FLASH_OB_Launch (void)
Function Description	Launch the option byte loading.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status

16.3.17 HAL_FLASH_GetError

Function Name	uint32_t HAL_FLASH_GetError (void)
Function Description	Get the specific FLASH error flag.

Return values

- FLASH_ErrorCode The returned value can be:
 HAL_FLASH_ERROR_WRP: FLASH Write protected error flag
 HAL_FLASH_ERROR_PGA: FLASH Programming Alignment error flag
 HAL_FLASH_ERROR_SIZE: FLASH Size error flag
 HAL_FLASH_ERROR_OPTV: Option validity error flag
 HAL_FLASH_ERROR_OPTVUSR: Option UserValidity Error flag (available only Cat.3, Cat.4 and Cat.5 devices)
 HAL_FLASH_ERROR_RD: FLASH Read Protection error flag (PCROP) (available only Cat.2 and Cat.3 devices)

16.4 FLASH Firmware driver defines

16.4.1 FLASH

FLASH Error Codes

HAL_FLASH_ERROR_NONE
 HAL_FLASH_ERROR_SIZE
 HAL_FLASH_ERROR_OPTV
 HAL_FLASH_ERROR_OPTVUSR
 HAL_FLASH_ERROR_PGA
 HAL_FLASH_ERROR_WRP
 HAL_FLASH_ERROR_RD
 HAL_FLASH_ERROR_OPERATION

FLASH Flags

FLASH_FLAG_BSY	FLASH Busy flag
FLASH_FLAG_EOP	FLASH End of Programming flag
FLASH_FLAG_ENDHV	FLASH End of High Voltage flag
FLASH_FLAG_READY	FLASH Ready flag after low power mode
FLASH_FLAG_WRPERR	FLASH Write protected error flag
FLASH_FLAG_PGAERR	FLASH Programming Alignment error flag
FLASH_FLAG_SIZERR	FLASH Size error flag
FLASH_FLAG_OPTVERR	FLASH Option Validity error flag

FLASH Interrupts

__HAL_FLASH_ENABLE_IT

Description:

- Enable the specified FLASH interrupt.

Parameters:

- __INTERRUPT__: : FLASH interrupt This parameter can be any combination of the following values:
 - FLASH_IT_EOP: End of FLASH Operation Interrupt
 - FLASH_IT_ERR: Error Interrupt

Return value:

__HAL_FLASH_DISABLE_IT

- none

Description:

- Disable the specified FLASH interrupt.

Parameters:

- **__INTERRUPT__**: FLASH interrupt This parameter can be any combination of the following values:
 - FLASH_IT_EOP: End of FLASH Operation Interrupt
 - FLASH_IT_ERR: Error Interrupt

Return value:

- none

__HAL_FLASH_GET_FLAG**Description:**

- Get the specified FLASH flag status.

Parameters:

- **__FLAG__**: specifies the FLASH flag to check. This parameter can be one of the following values:
 - FLASH_FLAG_BSY : FLASH Busy flag
 - FLASH_FLAG_EOP : FLASH End of Operation flag
 - FLASH_FLAG_ENDHV : FLASH End of High Voltage flag
 - FLASH_FLAG_READY: FLASH Ready flag after low power mode
 - FLASH_FLAG_WRPERR: FLASH Write protected error flag
 - FLASH_FLAG_PGAERR: FLASH Programming Alignment error flag
 - FLASH_FLAG_SIZERR: FLASH Size error flag
 - FLASH_FLAG_OPTVERR: FLASH Option validity error error flag
 - FLASH_FLAG_OPTVERRUSR : FLASH Option UserValidity (available only Cat.3, Cat.4 and Cat.5 devices)
 - FLASH_FLAG_RDERR : FLASH Read Protection error flag (PCROP) (available only Cat.2 and Cat.3 devices)

Return value:

- The: new state of **__FLAG__** (SET or RESET).

__HAL_FLASH_CLEAR_FLAG**Description:**

- Clear the specified FLASH flag.

Parameters:

- **__FLAG__**: specifies the FLASH flags to clear. This parameter can be any combination of the following values:
 - FLASH_FLAG_BSY : FLASH Busy flag
 - FLASH_FLAG_EOP : FLASH End of Operation

- flag
- FLASH_FLAG_ENDHV : FLASH End of High Voltage flag
- FLASH_FLAG_READY: FLASH Ready flag after low power mode
- FLASH_FLAG_WRPERR: FLASH Write protected error flag
- FLASH_FLAG_PGAERR: FLASH Programming Alignment error flag
- FLASH_FLAG_SIZERR: FLASH Size error flag
- FLASH_FLAG_OPTVERR: FLASH Option validity error error flag
- FLASH_FLAG_OPTVERRUSR : FLASH Option UserValidity (available only Cat.3, Cat.4 and Cat.5 devices)
- FLASH_FLAG_RDERR : FLASH Read Protection error flag (PCROP) (available only Cat.2 and Cat.3 devices)

Return value:

- none

FLASH Interrupts

FLASH_IT_EOP End of programming interrupt source

FLASH_IT_ERR Error interrupt source

FLASH Keys

FLASH_PDKEY1 Flash power down key1

FLASH_PDKEY2 Flash power down key2: used with FLASH_PDKEY1 to unlock the RUN_PD bit in FLASH_ACR

FLASH_PEKEY1 Flash program erase key1

FLASH_PEKEY2 Flash program erase key: used with FLASH_PEKEY2 to unlock the write access to the FLASH_PECR register and data EEPROM

FLASH_PRGKEY1 Flash program memory key1

FLASH_PRGKEY2 Flash program memory key2: used with FLASH_PRGKEY2 to unlock the program memory

FLASH_OPTKEY1 Flash option key1

FLASH_OPTKEY2 Flash option key2: used with FLASH_OPTKEY1 to unlock the write access to the option byte block

FLASH Latency

FLASH_LATENCY_0 FLASH Zero Latency cycle

FLASH_LATENCY_1 FLASH One Latency cycle

Private Define

FLASH_TIMEOUT_VALUE

FLASH_PAGE_SIZE

Private Macros

IS_FLASH_TYPEPROGRAM

IS_FLASH_LATENCY

FLASH Type Program

FLASH_TYPEPROGRAM_WORD Program a word (32-bit) at a specified address

17 HAL FLASH Extension Driver

17.1 HAL FLASH Extension Driver

17.2 FLASHEX Firmware driver registers structures

17.2.1 FLASH_EraseInitTypeDef

Data Fields

- *uint32_t TypeErase*
- *uint32_t PageAddress*
- *uint32_t NbPages*

Field Documentation

- *uint32_t FLASH_EraseInitTypeDef::TypeErase*
TypeErase: Page Erase only. This parameter can be a value of [FLASHEX_Type_Erase](#)
- *uint32_t FLASH_EraseInitTypeDef::PageAddress*
PageAddress: Initial FLASH address to be erased This parameter must be a value belonging to FLASH Programm address (depending on the devices)
- *uint32_t FLASH_EraseInitTypeDef::NbPages*
NbPages: Number of pages to be erased. This parameter must be a value between 1 and (max number of pages - value of Initial page)

17.2.2 FLASH_OBProgramInitTypeDef

Data Fields

- *uint32_t OptionType*
- *uint32_t WRPState*
- *uint32_t WRPSector0To31*
- *uint32_t WRPSector32To63*
- *uint32_t WRPSector64To95*
- *uint8_t RDPLLevel*
- *uint8_t BORLevel*
- *uint8_t USERConfig*

Field Documentation

- *uint32_t FLASH_OBProgramInitTypeDef::OptionType*
OptionType: Option byte to be configured. This parameter can be a value of [FLASHEX_Option_Type](#)
- *uint32_t FLASH_OBProgramInitTypeDef::WRPState*
WRPState: Write protection activation or deactivation. This parameter can be a value of [FLASHEX_WRP_State](#)

- ***uint32_t FLASH_OBProgramInitTypeDef::WRPSector0To31***
WRPSector0To31: specifies the sector(s) which are write protected between Sectors 0 to 31 This parameter can be a combination of [FLASHEx_Option_Bytes_Write_Protection1](#)
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPSector32To63***
WRPSector32To63: specifies the sector(s) which are write protected between Sectors 32 to 63 or Sectors 32 to 47 for STM32L1xxxDX devices. This parameter can be a combination of [FLASHEx_Option_Bytes_Write_Protection2](#)
- ***uint32_t FLASH_OBProgramInitTypeDef::WRPSector64To95***
WRPSector64to95: specifies the sector(s) which are write protected between Sectors 64 to 95 This parameter can be a combination of [FLASHEx_Option_Bytes_Write_Protection3](#)
- ***uint8_t FLASH_OBProgramInitTypeDef::RDPLLevel***
RDPLLevel: Set the read protection level.. This parameter can be a value of [FLASHEx_Option_Bytes_Read_Protection](#)
- ***uint8_t FLASH_OBProgramInitTypeDef::BORLevel***
BORLevel: Set the BOR Level. This parameter can be a value of [FLASHEx_Option_Bytes_BOR_Level](#)
- ***uint8_t FLASH_OBProgramInitTypeDef::USERConfig***
USERConfig: Program the FLASH User Option Byte: IWDG_SW / RST_STOP / RST_STDBY. This parameter can be a combination of [FLASHEx_Option_Bytes_IWatchdog](#), [FLASHEx_Option_Bytes_nRST_STOP](#) and [FLASHEx_Option_Bytes_nRST_STDBY](#)

17.2.3 FLASH_AdvOBProgramInitTypeDef

Data Fields

- ***uint32_t OptionType***
- ***uint16_t BootConfig***

Field Documentation

- ***uint32_t FLASH_AdvOBProgramInitTypeDef::OptionType***
OptionType: Option byte to be configured for extension . This parameter can be a value of [FLASHEx_OptionAdv_Type](#)
- ***uint16_t FLASH_AdvOBProgramInitTypeDef::BootConfig***
BootConfig: specifies Option bytes for boot config This parameter can be a value of [FLASHEx_Option_Bytes_BOOT](#)

17.3 FLASHEx Firmware driver API description

17.3.1 FLASH Erasing Programming functions

The FLASH Memory Erasing functions, includes the following functions:

- HAL_FLASHEx_Erase: return only when erase has been done
- HAL_FLASHEx_Erase_IT: end of erase is done when HAL_FLASH_EndOfOperationCallback is called with parameter 0xFFFFFFFF

Any operation of erase should follow these steps:

1. Call the HAL_FLASH_Unlock() function to enable the flash control register and program memory access.
2. Call the desired function to erase page.
3. Call the HAL_FLASH_Lock() to disable the flash program memory access (recommended to protect the FLASH memory against possible unwanted operation).

This section contains the following APIs:

- [HAL_FLASHEx_Erase\(\)](#)
- [HAL_FLASHEx_Erase_IT\(\)](#)

17.3.2 Option Bytes Programming functions

Any operation of erase or program should follow these steps:

1. Call the HAL_FLASH_OB_Unlock() function to enable the Flash option control register access.
2. Call following function to program the desired option bytes.
 - HAL_FLASHEx_OBProgram: - To Enable/Disable the desired sector write protection. - To set the desired read Protection Level. - To configure the user option Bytes: IWDG, STOP and the Standby. - To Set the BOR level.
3. Once all needed option bytes to be programmed are correctly written, call the HAL_FLASH_OB_Launch(void) function to launch the Option Bytes programming process.
4. Call the HAL_FLASH_OB_Lock() to disable the Flash option control register access (recommended to protect the option Bytes against possible unwanted operations).

Proprietary code Read Out Protection (PcROP):

1. The PcROP sector is selected by using the same option bytes as the Write protection (nWRPi bits). As a result, these 2 options are exclusive each other.
2. In order to activate the PcROP (change the function of the nWRPi option bits), the SPRMOD option bit must be activated.
3. The active value of nWRPi bits is inverted when PCROP mode is active, this means: if SPRMOD = 1 and nWRPi = 1 (default value), then the user sector "i" is read/write protected.
4. To activate PCROP mode for Flash sector(s), you need to call the following function:
 - HAL_FLASHEx_AdvOBProgram in selecting sectors to be read/write protected
 - HAL_FLASHEx_OB_SelectPCROP to enable the read/write protection
5. PcROP is available only in STM32L151xBA, STM32L152xBA, STM32L151xC, STM32L152xC & STM32L162xC devices.

This section contains the following APIs:

- [HAL_FLASHEx_OBProgram\(\)](#)
- [HAL_FLASHEx_OBGetConfig\(\)](#)
- [HAL_FLASHEx_AdvOBProgram\(\)](#)
- [HAL_FLASHEx_AdvOBGetConfig\(\)](#)

17.3.3 DATA EEPROM Programming functions

Any operation of erase or program should follow these steps:

1. Call the HAL_FLASHEx_DATAEEPROM_Unlock() function to enable the data EEPROM access and Flash program erase control register access.
2. Call the desired function to erase or program data.
3. Call the HAL_FLASHEx_DATAEEPROM_Lock() to disable the data EEPROM access and Flash program erase control register access (recommended to protect the DATA_EEPROM against possible unwanted operation).

This section contains the following APIs:

- [HAL_FLASHEx_DATAEEPROM_Unlock\(\)](#)
- [HAL_FLASHEx_DATAEEPROM_Lock\(\)](#)
- [HAL_FLASHEx_DATAEEPROM_Erase\(\)](#)
- [HAL_FLASHEx_DATAEEPROM_Program\(\)](#)
- [HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram\(\)](#)
- [HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram\(\)](#)

17.3.4 HAL_FLASHEx_Erase

Function Name	HAL_StatusTypeDef HAL_FLASHEx_Erase (FLASH_EraseInitTypeDef * pEraseInit, uint32_t * PageError)
Function Description	Erase the specified FLASH memory Pages.
Parameters	<ul style="list-style-type: none"> • pEraseInit: pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing. • PageError: pointer to variable that contains the configuration information on faulty sector in case of error (0xFFFFFFFF means that all the sectors have been correctly erased)
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> • To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation) • For STM32L151xDX/STM32L152xDX/STM32L162xDX, as memory is not continuous between 2 banks, user should perform pages erase by bank only.

17.3.5 HAL_FLASHEx_Erase_IT

Function Name	HAL_StatusTypeDef HAL_FLASHEx_Erase_IT (FLASH_EraseInitTypeDef * pEraseInit)
Function Description	Perform a page erase of the specified FLASH memory pages with interrupt enabled.
Parameters	<ul style="list-style-type: none"> • pEraseInit: pointer to an FLASH_EraseInitTypeDef structure that contains the configuration information for the erasing.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> • To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation) • For STM32L151xDX/STM32L152xDX/STM32L162xDX, as memory is not continuous between 2 banks, user should perform pages erase by bank only.

17.3.6 HAL_FLASHEx_OBProgram

Function Name	HAL_StatusTypeDef HAL_FLASHEx_OBProgram (FLASH_OBProgramInitTypeDef * pOBInit)
Function Description	Program option bytes.

Parameters	<ul style="list-style-type: none"> • pOBInit: pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status

17.3.7 HAL_FLASHEx_OBGetConfig

Function Name	void HAL_FLASHEx_OBGetConfig (FLASH_OBProgramInitTypeDef * pOBInit)
Function Description	Get the Option byte configuration.
Parameters	<ul style="list-style-type: none"> • pOBInit: pointer to an FLASH_OBInitStruct structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none"> • None

17.3.8 HAL_FLASHEx_AdvOBProgram

Function Name	HAL_StatusTypeDef HAL_FLASHEx_AdvOBProgram (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)
Function Description	Program option bytes.
Parameters	<ul style="list-style-type: none"> • pAdvOBInit: pointer to an FLASH_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status
Notes	<ul style="list-style-type: none"> • This function can be used only for Cat2 & Cat3 devices for PCROP and Cat4 & Cat5 for BFB2.

17.3.9 HAL_FLASHEx_AdvOBGetConfig

Function Name	void HAL_FLASHEx_AdvOBGetConfig (FLASH_AdvOBProgramInitTypeDef * pAdvOBInit)
Function Description	Get the OBEX byte configuration.
Parameters	<ul style="list-style-type: none"> • pAdvOBInit: pointer to an FLASH_AdvOBProgramInitTypeDef structure that contains the configuration information for the programming.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This function can be used only for Cat2 & Cat3 devices for PCROP and Cat4 & Cat5 for BFB2.

17.3.10 HAL_FLASHEx_DATAEEPROM_Unlock

Function Name	HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Unlock (void)
Function Description	Unlocks the data memory and FLASH_PECR register access.
Return values	<ul style="list-style-type: none"> • HAL_StatusTypeDef HAL Status

17.3.11 HAL_FLASHEx_DATAEEPROM_Lock

Function Name	HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Lock
---------------	--

(void)

Function Description Locks the Data memory and FLASH_PECR register access.

Return values

- HAL_StatusTypeDef HAL Status

17.3.12 HAL_FLASHEx_DATAEEPROM_Erase

Function Name **HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Erase (uint32_t TypeErase, uint32_t Address)**

Function Description Erase a word in data memory.

Parameters

- **Address:** specifies the address to be erased.
- **TypeErase:** Indicate the way to erase at a specified address. This parameter can be a value of FLASH Type Program

Return values

- FLASH Status: The returned value can be: FLASH_ERROR_PROGRAM, HAL_FLASH_ERROR_WRP, FLASH_COMPLETE or FLASH_TIMEOUT.

Notes

- To correctly run this function, the DATA_EEPROM_Unlock() function must be called before. Call the DATA_EEPROM_Lock() to the data EEPROM access and Flash program erase control register access(recommended to protect the DATA_EEPROM against possible unwanted operation).

17.3.13 HAL_FLASHEx_DATAEEPROM_Program

Function Name **HAL_StatusTypeDef HAL_FLASHEx_DATAEEPROM_Program (uint32_t TypeProgram, uint32_t Address, uint32_t Data)**

Function Description Program word at a specified address.

Parameters

- **TypeProgram:** Indicate the way to program at a specified address. This parameter can be a value of FLASHEx Type Program Data
- **Address:** specifies the address to be programmed.
- **Data:** specifies the data to be programmed

Return values

- HAL_StatusTypeDef HAL Status

Notes

- To correctly run this function, the HAL_FLASH_EEPROM_Unlock() function must be called before. Call the HAL_FLASHEx_DATAEEPROM_Unlock() to the data EEPROM access and Flash program erase control register access(recommended to protect the DATA_EEPROM against possible unwanted operation).
- The function HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram() can be called before this function to configure the Fixed Time Programming.

17.3.14 HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram

Function Name **void HAL_FLASHEx_DATAEEPROM_EnableFixedTimeProgram (void)**

Function Description	Enable DATA EEPROM fixed Time programming (2*Tprog).
Return values	<ul style="list-style-type: none"> • None

17.3.15 HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram

Function Name	void HAL_FLASHEx_DATAEEPROM_DisableFixedTimeProgram (void)
Function Description	Disables DATA EEPROM fixed Time programming (2*Tprog).
Return values	<ul style="list-style-type: none"> • None

17.4 FLASHEx Firmware driver defines

17.4.1 FLASHEx

FLASHEx Address

IS_FLASH_DATA_ADDRESS
 IS_FLASH_PROGRAM_ADDRESS
 IS_FLASH_PROGRAM_BANK1_ADDRESS
 IS_FLASH_PROGRAM_BANK2_ADDRESS
 IS_NBPAGES
 IS_OB_BOOT_BANK

Exported Macros

__HAL_FLASH_SET_LATENCY

Description:

- Set the FLASH Latency.

Parameters:

- **__LATENCY__**: FLASH Latency This parameter can be one of the following values:
 - **FLASH_LATENCY_0**: FLASH Zero Latency cycle
 - **FLASH_LATENCY_1**: FLASH One Latency cycle

Return value:

- none

__HAL_FLASH_GET_LATENCY

Description:

- Get the FLASH Latency.

Return value:

- **FLASH**: Latency This parameter can be one of the following values:
 - **FLASH_LATENCY_0**: FLASH Zero Latency cycle
 - **FLASH_LATENCY_1**:

FLASH One Latency cycle

`__HAL_FLASH_ACC64_ENABLE`**Description:**

- Enable the FLASH 64-bit access.

Return value:

- none

Notes:

- Read access 64 bit is used. This bit cannot be written at the same time as the LATENCY and PRFTEN bits.

`__HAL_FLASH_ACC64_DISABLE`**Description:**

- Disable the FLASH 64-bit access.

Return value:

- none

Notes:

- Read access 32 bit is used To reset this bit, the LATENCY should be zero wait state and the prefetch off.

`__HAL_FLASH_PREFETCH_BUFFER_ENABLE`**Description:**

- Enable the FLASH prefetch buffer.

Return value:

- none

`__HAL_FLASH_PREFETCH_BUFFER_DISABLE`**Description:**

- Disable the FLASH prefetch buffer.

Return value:

- none

`__HAL_FLASH_SLEEP_POWERDOWN_ENABLE`**Description:**

- Enable the FLASH power down during Sleep mode.

Return value:

- none

`__HAL_FLASH_SLEEP_POWERDOWN_DISABLE`**Description:**

- Disable the FLASH power down during Sleep mode.

Return value:

- none

__HAL_FLASH_POWER_DOWN_ENABLE

Notes:

- Writing this bit to 0 this bit, automatically the keys are loss and a new unlock sequence is necessary to re-write it to 1.

__HAL_FLASH_POWER_DOWN_DISABLE

FLASHEx Flags

FLASH_FLAG_OPTVERRUSR FLASH Option User Validity error flag

FLASHEx Option Advanced Type

OPTIONBYTE_BOOTCONFIG BOOTConfig option byte configuration

FLASHEx Option Bytes BOOT

OB_BOOT_BANK2 At startup, if boot pins are set in boot from user Flash position and this parameter is selected the device will boot from Bank 2 or Bank 1, depending on the activation of the bank

OB_BOOT_BANK1 At startup, if boot pins are set in boot from user Flash position and this parameter is selected the device will boot from Bank1(Default)

FLASHEx Option Bytes BOR Level

OB_BOR_OFF BOR is disabled at power down, the reset is asserted when the VDD power supply reaches the PDR(Power Down Reset) threshold (1.5V)

OB_BOR_LEVEL1 BOR Reset threshold levels for 1.7V - 1.8V VDD power supply

OB_BOR_LEVEL2 BOR Reset threshold levels for 1.9V - 2.0V VDD power supply

OB_BOR_LEVEL3 BOR Reset threshold levels for 2.3V - 2.4V VDD power supply

OB_BOR_LEVEL4 BOR Reset threshold levels for 2.55V - 2.65V VDD power supply

OB_BOR_LEVEL5 BOR Reset threshold levels for 2.8V - 2.9V VDD power supply

FLASHEx Option Bytes IWatchdog

OB_IWDG_SW Software WDG selected

OB_IWDG_HW Hardware WDG selected

FLASHEx Option Bytes nRST_STDBY

OB_STDBY_NORST No reset generated when entering in STANDBY

OB_STDBY_RST Reset generated when entering in STANDBY

FLASHEx Option Bytes nRST_STOP

OB_STOP_NORST No reset generated when entering in STOP

OB_STOP_RST Reset generated when entering in STOP

FLASHEx Option Bytes Read Protection

OB_RDP_LEVEL0

OB_RDP_LEVEL1

FLASHEx Option Bytes Write Mask

WRP_MASK_LOW

WRP_MASK_HIGH

FLASHEx Option Bytes Write Protection1

OB_WRP1_PAGES0TO15

OB_WRP1_PAGES16TO31

OB_WRP1_PAGES32TO47

OB_WRP1_PAGES48TO63

OB_WRP1_PAGES64TO79

OB_WRP1_PAGES80TO95

OB_WRP1_PAGES96TO111

OB_WRP1_PAGES112TO127

OB_WRP1_PAGES128TO143

OB_WRP1_PAGES144TO159

OB_WRP1_PAGES160TO175

OB_WRP1_PAGES176TO191

OB_WRP1_PAGES192TO207

OB_WRP1_PAGES208TO223

OB_WRP1_PAGES224TO239

OB_WRP1_PAGES240TO255

OB_WRP1_PAGES256TO271

OB_WRP1_PAGES272TO287

OB_WRP1_PAGES288TO303

OB_WRP1_PAGES304TO319

OB_WRP1_PAGES320TO335

OB_WRP1_PAGES336TO351

OB_WRP1_PAGES352TO367

OB_WRP1_PAGES368TO383

OB_WRP1_PAGES384TO399

OB_WRP1_PAGES400TO415

OB_WRP1_PAGES416TO431

OB_WRP1_PAGES432TO447

OB_WRP1_PAGES448TO463

OB_WRP1_PAGES464TO479

OB_WRP1_PAGES480TO495

OB_WRP1_PAGES496TO511

OB_WRP1_ALLPAGES Write protection of all Sectors

FLASHEx Option Bytes Write Protection2

OB_WRP2_PAGES512TO527
OB_WRP2_PAGES528TO543
OB_WRP2_PAGES544TO559
OB_WRP2_PAGES560TO575
OB_WRP2_PAGES576TO591
OB_WRP2_PAGES592TO607
OB_WRP2_PAGES608TO623
OB_WRP2_PAGES624TO639
OB_WRP2_PAGES640TO655
OB_WRP2_PAGES656TO671
OB_WRP2_PAGES672TO687
OB_WRP2_PAGES688TO703
OB_WRP2_PAGES704TO719
OB_WRP2_PAGES720TO735
OB_WRP2_PAGES736TO751
OB_WRP2_PAGES752TO767
OB_WRP2_PAGES768TO783
OB_WRP2_PAGES784TO799
OB_WRP2_PAGES800TO815
OB_WRP2_PAGES816TO831
OB_WRP2_PAGES832TO847
OB_WRP2_PAGES848TO863
OB_WRP2_PAGES864TO879
OB_WRP2_PAGES880TO895
OB_WRP2_PAGES896TO911
OB_WRP2_PAGES912TO927
OB_WRP2_PAGES928TO943
OB_WRP2_PAGES944TO959
OB_WRP2_PAGES960TO975
OB_WRP2_PAGES976TO991
OB_WRP2_PAGES992TO1007
OB_WRP2_PAGES1008TO1023

OB_WRP2_ALLPAGES Write protection of all Sectors

FLASHEx Option Bytes Write Protection3

OB_WRP3_PAGES1024TO1039
OB_WRP3_PAGES1040TO1055

OB_WRP3_PAGES1056TO1071
OB_WRP3_PAGES1072TO1087
OB_WRP3_PAGES1088TO1103
OB_WRP3_PAGES1104TO1119
OB_WRP3_PAGES1120TO1135
OB_WRP3_PAGES1136TO1151
OB_WRP3_PAGES1152TO1167
OB_WRP3_PAGES1168TO1183
OB_WRP3_PAGES1184TO1199
OB_WRP3_PAGES1200TO1215
OB_WRP3_PAGES1216TO1231
OB_WRP3_PAGES1232TO1247
OB_WRP3_PAGES1248TO1263
OB_WRP3_PAGES1264TO1279
OB_WRP3_PAGES1280TO1295
OB_WRP3_PAGES1296TO1311
OB_WRP3_PAGES1312TO1327
OB_WRP3_PAGES1328TO1343
OB_WRP3_PAGES1344TO1359
OB_WRP3_PAGES1360TO1375
OB_WRP3_PAGES1376TO1391
OB_WRP3_PAGES1392TO1407
OB_WRP3_PAGES1408TO1423
OB_WRP3_PAGES1424TO1439
OB_WRP3_PAGES1440TO1455
OB_WRP3_PAGES1456TO1471
OB_WRP3_PAGES1472TO1487
OB_WRP3_PAGES1488TO1503
OB_WRP3_PAGES1504TO1519
OB_WRP3_PAGES1520TO1535

OB_WRP3_ALLPAGES Write protection of all Sectors

FLASHEx Option Type

OPTIONBYTE_WRP WRP option byte configuration
OPTIONBYTE_RDP RDP option byte configuration
OPTIONBYTE_USER USER option byte configuration
OPTIONBYTE_BOR BOR option byte configuration

Private Defines

FLASH_FLAG_MASK

FLASH_NBPAGES_MAX

Private Macros

IS_FLASH_TYPEERASE

IS_OPTIONBYTE

IS_WRPSTATE

IS_OB_RDP

IS_OB_BOR_LEVEL

IS_OB_IWDG_SOURCE

IS_OB_STOP_SOURCE

IS_OB_STDBY_SOURCE

IS_OBEX

IS_TYPEERASEDATA

IS_TYPEPROGRAMDATA

FLASHEx_Type_Erase

FLASH_TYPEERASE_PAGES Page erase only

FLASHEx_Type_Erase Data

FLASH_TYPEERASEDATA_BYTE Erase byte (8-bit) at a specified address.

FLASH_TYPEERASEDATA_HALFWORD Erase a half-word (16-bit) at a specified address.

FLASH_TYPEERASEDATA_WORD Erase a word (32-bit) at a specified address.

FLASHEx_Type_Program Data

FLASH_TYPEPROGRAMDATA_BYTE Program byte (8-bit) at a specified address.

FLASH_TYPEPROGRAMDATA_HALFWORD Program a half-word (16-bit) at a specified address.

FLASH_TYPEPROGRAMDATA_WORD Program a word (32-bit) at a specified address.

FLASH_TYPEPROGRAMDATA_FASTBYTE Fast Program byte (8-bit) at a specified address.

FLASH_TYPEPROGRAMDATA_FASTHALFWORD Fast Program a half-word (16-bit) at a specified address.

FLASH_TYPEPROGRAMDATA_FASTWORD Fast Program a word (32-bit) at a specified address.

FLASHEx_WRP State

OB_WRPSTATE_DISABLE Disable the write protection of the desired bank 1 sectors

OB_WRPSTATE_ENABLE Enable the write protection of the desired bank 1 sectors

18 HAL FLASH__RAMFUNC Generic Driver

18.1 HAL FLASH__RAMFUNC Generic Driver

18.2 FLASH__RAMFUNC Firmware driver API description

18.2.1 HAL_FLASHEx_EnableRunPowerDown

Function Name	<code>__RAM_FUNC HAL_FLASHEx_EnableRunPowerDown (void)</code>
Function Description	Enable the power down mode during RUN mode.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> This function can be used only when the user code is running from Internal SRAM.

18.2.2 HAL_FLASHEx_DisableRunPowerDown

Function Name	<code>__RAM_FUNC HAL_FLASHEx_DisableRunPowerDown (void)</code>
Function Description	Disable the power down mode during RUN mode.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> This function can be used only when the user code is running from Internal SRAM.

18.2.3 HAL_FLASHEx_EraseParallelPage

Function Name	<code>__RAM_FUNC HAL_FLASHEx_EraseParallelPage (uint32_t Page_Address1, uint32_t Page_Address2)</code>
Function Description	Erases a specified 2 page in program memory in parallel.
Parameters	<ul style="list-style-type: none"> Page_Address1: The page address in program memory to be erased in the first Bank (BANK1). This parameter should be between FLASH_BASE and FLASH_BANK1_END. Page_Address2: The page address in program memory to be erased in the second Bank (BANK2). This parameter should be between FLASH_BANK2_BASE and FLASH_BANK2_END.
Return values	<ul style="list-style-type: none"> HAL Status: The returned value can be: HAL_ERROR, HAL_OK or HAL_TIMEOUT.
Notes	<ul style="list-style-type: none"> This function can be used only for STM32L151xD, STM32L152xD), STM32L162xD and Cat5 devices. To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation). A Page is erased in the Program memory only if the address to load is the start address of a page (multiple of 256 bytes).

18.2.4 HAL_FLASHEx_ProgramParallelHalfPage



Function Name	__RAM_FUNC HAL_FLASHEx_ProgramParallelHalfPage (uint32_t Address1, uint32_t * pBuffer1, uint32_t Address2, uint32_t * pBuffer2)
Function Description	Programs 2 half page in program memory in parallel.
Parameters	<ul style="list-style-type: none"> • Address1: specifies the first address to be written in the first bank (BANK1). This parameter should be between FLASH_BASE and (FLASH_BANK1_END - FLASH_PAGE_SIZE). • pBuffer1: pointer to the buffer containing the data to be written to the first half page in the first bank. • Address2: specifies the second address to be written in the second bank (BANK2). This parameter should be between FLASH_BANK2_BASE and (FLASH_BANK2_END - FLASH_PAGE_SIZE). • pBuffer2: pointer to the buffer containing the data to be written to the second half page in the second bank.
Return values	<ul style="list-style-type: none"> • HAL Status: The returned value can be: HAL_ERROR, HAL_OK or HAL_TIMEOUT.
Notes	<ul style="list-style-type: none"> • This function can be used only for STM32L151xD, STM32L152xD, STM32L162xD and Cat5 devices. • To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation). • Half page write is possible only from SRAM. • If there are more than 32 words to write, after 32 words another Half Page programming operation starts and has to be finished. • A half page is written to the program memory only if the first address to load is the start address of a half page (multiple of 128 bytes) and the 31 remaining words to load are in the same half page. • During the Program memory half page write all read operations are forbidden (this includes DMA read operations and debugger read operations such as breakpoints, periodic updates, etc.). • If a PGAERR is set during a Program memory half page write, the complete write operation is aborted. Software should then reset the FPRG and PROG/DATA bits and restart the write operation from the beginning.

18.2.5 HAL_FLASHEx_HalfPageProgram

Function Name	__RAM_FUNC HAL_FLASHEx_HalfPageProgram (uint32_t Address, uint32_t * pBuffer)
Function Description	Programs a half page in program memory.
Parameters	<ul style="list-style-type: none"> • Address: specifies the address to be written. • pBuffer: pointer to the buffer containing the data to be written to the half page.
Return values	<ul style="list-style-type: none"> • HAL Status: The returned value can be: HAL_ERROR,

Notes

HAL_OK or HAL_TIMEOUT.

- To correctly run this function, the HAL_FLASH_Unlock() function must be called before. Call the HAL_FLASH_Lock() to disable the flash memory access (recommended to protect the FLASH memory against possible unwanted operation)
- Half page write is possible only from SRAM.
- If there are more than 32 words to write, after 32 words another Half Page programming operation starts and has to be finished.
- A half page is written to the program memory only if the first address to load is the start address of a half page (multiple of 128 bytes) and the 31 remaining words to load are in the same half page.
- During the Program memory half page write all read operations are forbidden (this includes DMA read operations and debugger read operations such as breakpoints, periodic updates, etc.).
- If a PGAERR is set during a Program memory half page write, the complete write operation is aborted. Software should then reset the FPRG and PROG/DATA bits and restart the write operation from the beginning.

18.2.6 HAL_FLASHEx_DATAEEPROM_EraseDoubleWord

Function Name

__RAM_FUNC
HAL_FLASHEx_DATAEEPROM_EraseDoubleWord (uint32_t Address)

Function Description

Erase a double word in data memory.

Parameters

- **Address:** specifies the address to be erased.

Return values

- HAL Status: The returned value can be: HAL_ERROR, HAL_OK or HAL_TIMEOUT.

Notes

- To correctly run this function, the HAL_FLASH_EEPROM_Unlock() function must be called before. Call the HAL_FLASH_EEPROM_Lock() to the data EEPROM access and Flash program erase control register access (recommended to protect the DATA_EEPROM against possible unwanted operation).
- Data memory double word erase is possible only from SRAM.
- A double word is erased to the data memory only if the first address to load is the start address of a double word (multiple of 8 bytes).
- During the Data memory double word erase, all read operations are forbidden (this includes DMA read operations and debugger read operations such as breakpoints, periodic updates, etc.).

18.2.7 HAL_FLASHEx_DATAEEPROM_ProgramDoubleWord

Function Name

__RAM_FUNC
HAL_FLASHEx_DATAEEPROM_ProgramDoubleWord (uint32_t Address, uint64_t Data)

Function Description	Write a double word in data memory without erase.
Parameters	<ul style="list-style-type: none">• Address: specifies the address to be written.• Data: specifies the data to be written.
Return values	<ul style="list-style-type: none">• HAL Status: The returned value can be: HAL_ERROR, HAL_OK or HAL_TIMEOUT.
Notes	<ul style="list-style-type: none">• To correctly run this function, the HAL_FLASH_EEPROM_Unlock() function must be called before. Call the HAL_FLASH_EEPROM_Lock() to he data EEPROM access and Flash program erase control register access(recommended to protect the DATA_EEPROM against possible unwanted operation).• Data memory double word write is possible only from SRAM.• A data memory double word is written to the data memory only if the first address to load is the start address of a double word (multiple of double word).• During the Data memory double word write, all read operations are forbidden (this includes DMA read operations and debugger read operations such as breakpoints, periodic updates, etc.).

19 HAL GPIO Generic Driver

19.1 HAL GPIO Generic Driver

19.2 GPIO Firmware driver registers structures

19.2.1 GPIO_InitTypeDef

Data Fields

- *uint32_t Pin*
- *uint32_t Mode*
- *uint32_t Pull*
- *uint32_t Speed*
- *uint32_t Alternate*

Field Documentation

- *uint32_t GPIO_InitTypeDef::Pin*
Specifies the GPIO pins to be configured. This parameter can be any value of [GPIO_pins](#)
- *uint32_t GPIO_InitTypeDef::Mode*
Specifies the operating mode for the selected pins. This parameter can be a value of [GPIO_mode](#)
- *uint32_t GPIO_InitTypeDef::Pull*
Specifies the Pull-up or Pull-Down activation for the selected pins. This parameter can be a value of [GPIO_pull](#)
- *uint32_t GPIO_InitTypeDef::Speed*
Specifies the speed for the selected pins. This parameter can be a value of [GPIO_speed](#)
- *uint32_t GPIO_InitTypeDef::Alternate*
Peripheral to be connected to the selected pins This parameter can be a value of [GPIOEx_Alternate_function_selection](#)

19.3 GPIO Firmware driver API description

19.3.1 GPIO Peripheral features

Each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:

- Input mode
- Analog mode
- Output mode
- Alternate function mode
- External interrupt/event lines

During and just after reset, the alternate functions and external interrupt lines are not active and the I/O ports are configured in input floating mode.

All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not.

In Output or Alternate mode, each IO can be configured on open-drain or push-pull type and the IO speed can be selected depending on the VDD value.

The microcontroller IO pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral's alternate function (AF) connected to an IO pin at a time. In this way, there can be no conflict between peripherals sharing the same IO pin.

All ports have external interrupt/event capability. To use external interrupt lines, the port must be configured in input mode. All available GPIO pins are connected to the 16 external interrupt/event lines from EXTI0 to EXTI15.

The external interrupt/event controller consists of up to 28 edge detectors (depending on products 16 lines are connected to GPIO) for generating event/interrupt requests (each input line can be independently configured to select the type (interrupt or event) and the corresponding trigger event (rising or falling or both). Each line can also be masked independently.

19.3.2 How to use this driver

1. Enable the GPIO AHB clock using the following function : `__GPIOx_CLK_ENABLE()`.
2. Configure the GPIO pin(s) using `HAL_GPIO_Init()`.
 - Configure the IO mode using "Mode" member from `GPIO_InitTypeDef` structure
 - Activate Pull-up, Pull-down resistor using "Pull" member from `GPIO_InitTypeDef` structure.
 - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from `GPIO_InitTypeDef` structure, the speed is configurable: Low, Medium and High.
 - If alternate mode is selected, the alternate function connected to the IO is configured through "Alternate" member from `GPIO_InitTypeDef` structure
 - Analog mode is required when a pin is to be used as ADC channel or DAC output.
 - In case of external interrupt/event selection the "Mode" member from `GPIO_InitTypeDef` structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using `HAL_NVIC_SetPriority()` and enable it using `HAL_NVIC_EnableIRQ()`.
4. `HAL_GPIO_DeInit` allows to set register values to their reset value. It's also recommended to use it to unconfigure pin which was used as an external interrupt or in event mode. That's the only way to reset corresponding bit in EXTI & SYSCFG registers.
5. To get the level of a pin configured in input mode use `HAL_GPIO_ReadPin()`.
6. To set/reset the level of a pin configured in output mode use `HAL_GPIO_WritePin()/HAL_GPIO_TogglePin()`.
7. To lock pin configuration until next reset use `HAL_GPIO_LockPin()`.
8. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
9. The LSE oscillator pins `OSC32_IN` and `OSC32_OUT` can be used as general purpose (`PC14` and `PC15`, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.

10. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

19.3.3 Initialization and Configuration functions

This section contains the following APIs:

- [HAL_GPIO_Init\(\)](#)
- [HAL_GPIO_DeInit\(\)](#)

19.3.4 HAL_GPIO_Init

Function Name	void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)
Function Description	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_Init.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..G depending on device used) to select the GPIO peripheral for STM32L1XX family devices • GPIO_Init: pointer to a GPIO_InitTypeDef structure that contains the configuration information for the specified GPIO peripheral.
Return values	<ul style="list-style-type: none"> • None

19.3.5 HAL_GPIO_DeInit

Function Name	void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)
Function Description	De-initializes the GPIOx peripheral registers to their default reset values.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..G depending on device used) to select the GPIO peripheral for STM32L1XX family devices • GPIO_Pin: specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15).
Return values	<ul style="list-style-type: none"> • None

19.3.6 HAL_GPIO_ReadPin

Function Name	GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
Function Description	Reads the specified input port pin.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..G depending on device used) to select the GPIO peripheral for STM32L1XX family devices • GPIO_Pin: specifies the port bit to read. This parameter can be GPIO_PIN_x where x can be (0..15).
Return values	<ul style="list-style-type: none"> • The input port pin value.

19.3.7 HAL_GPIO_WritePin

Function Name	void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
---------------	---

Function Description	Sets or clears the selected data port bit.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..G depending on device used) to select the GPIO peripheral for STM32L1XX family devices • GPIO_Pin: specifies the port bit to be written. This parameter can be one of GPIO_PIN_x where x can be (0..15). • PinState: specifies the value to be written to the selected bit. This parameter can be one of the GPIO_PinState enum values: GPIO_PIN_RESET: to clear the port pinGPIO_PIN_SET: to set the port pin
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This function uses GPIOx_BSRR register to allow atomic read/modify accesses. In this way, there is no risk of an IRQ occurring between the read and the modify access.

19.3.8 HAL_GPIO_TogglePin

Function Name	void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
Function Description	Toggles the specified GPIO pin.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..G depending on device used) to select the GPIO peripheral for STM32L1XX family devices • GPIO_Pin: specifies the pins to be toggled.
Return values	<ul style="list-style-type: none"> • None

19.3.9 HAL_GPIO_LockPin

Function Name	HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
Function Description	Locks GPIO Pins configuration registers.
Parameters	<ul style="list-style-type: none"> • GPIOx: where x can be (A..G depending on device used) to select the GPIO peripheral for STM32L1XX family devices • GPIO_Pin: Specifies the port bit to be locked. This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH. • The configuration of the locked GPIO pins can no longer be modified until the next reset. • Limitation concerning GPIOx_OTYPER: Locking of GPIOx_OTYPER[i] with i = 15..8 depends from setting of GPIOx_LCKR[i-8] and not from GPIOx_LCKR[i]. GPIOx_LCKR[i-8] is locking GPIOx_OTYPER[i] together with GPIOx_OTYPER[i-8]. It is not possible to lock GPIOx_OTYPER[i] with i = 15..8, without locking also GPIOx_OTYPER[i-8]. Workaround: When calling HAL_GPIO_LockPin with GPIO_Pin from GPIO_PIN_8 to GPIO_PIN_15, you must call also HAL_GPIO_LockPin with

GPIO_Pin - 8. (When locking a pin from GPIO_PIN_8 to GPIO_PIN_15, you must lock also the corresponding GPIO_PIN_0 to GPIO_PIN_7).

19.3.10 HAL_GPIO_EXTI_IRQHandler

Function Name	void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)
Function Description	This function handles EXTI interrupt request.
Parameters	<ul style="list-style-type: none"> GPIO_Pin: Specifies the port pin connected to corresponding EXTI line.
Return values	<ul style="list-style-type: none"> None

19.3.11 HAL_GPIO_EXTI_Callback

Function Name	void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)
Function Description	EXTI line detection callbacks.
Parameters	<ul style="list-style-type: none"> GPIO_Pin: Specifies the port pin connected to corresponding EXTI line.
Return values	<ul style="list-style-type: none"> None

19.4 GPIO Firmware driver defines

19.4.1 GPIO

GPIO Exported Macros

__HAL_GPIO_EXTI_GET_FLAG

Description:

- Checks whether the specified EXTI line flag is set or not.

Parameters:

- __EXTI_LINE__:** specifies the EXTI line flag to check. This parameter can be GPIO_PIN_x where x can be (0..15)

Return value:

- The: new state of __EXTI_LINE__ (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_FLAG

Description:

- Clears the EXTI's line pending flags.

Parameters:

- __EXTI_LINE__:** specifies the EXTI lines flags to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GET_IT**Description:**

- Checks whether the specified EXTI line is asserted or not.

Parameters:

- __EXTI_LINE__**: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- The: new state of **__EXTI_LINE__** (SET or RESET).

__HAL_GPIO_EXTI_CLEAR_IT**Description:**

- Clears the EXTI's line pending bits.

Parameters:

- __EXTI_LINE__**: specifies the EXTI lines to clear. This parameter can be any combination of GPIO_PIN_x where x can be (0..15)

Return value:

- None

__HAL_GPIO_EXTI_GENERATE_SWIT**Description:**

- Generates a Software interrupt on selected EXTI line.

Parameters:

- __EXTI_LINE__**: specifies the EXTI line to check. This parameter can be GPIO_PIN_x where x can be(0..15)

Return value:

- None

GPIO mode

GPIO_MODE_INPUT

Input Floating Mode

GPIO_MODE_OUTPUT_PP

Output Push Pull Mode

GPIO_MODE_OUTPUT_OD

Output Open Drain Mode

GPIO_MODE_AF_PP

Alternate Function Push Pull Mode

GPIO_MODE_AF_OD

Alternate Function Open Drain Mode

GPIO_MODE_ANALOG

Analog Mode

GPIO_MODE_IT_RISING

External Interrupt Mode with Rising edge trigger detection

GPIO_MODE_IT_FALLING

External Interrupt Mode with Falling edge trigger detection

GPIO_MODE_IT_RISING_FALLING

External Interrupt Mode with Rising/Falling edge

	trigger detection
GPIO_MODE_EVT_RISING	External Event Mode with Rising edge trigger detection
GPIO_MODE_EVT_FALLING	External Event Mode with Falling edge trigger detection
GPIO_MODE_EVT_RISING_FALLING	External Event Mode with Rising/Falling edge trigger detection

GPIO pins

GPIO_PIN_0
GPIO_PIN_1
GPIO_PIN_2
GPIO_PIN_3
GPIO_PIN_4
GPIO_PIN_5
GPIO_PIN_6
GPIO_PIN_7
GPIO_PIN_8
GPIO_PIN_9
GPIO_PIN_10
GPIO_PIN_11
GPIO_PIN_12
GPIO_PIN_13
GPIO_PIN_14
GPIO_PIN_15
GPIO_PIN_All
GPIO_PIN_MASK

GPIO Private Constants

GPIO_MODE
EXTI_MODE
GPIO_MODE_IT
GPIO_MODE_EVT
RISING_EDGE
FALLING_EDGE
GPIO_OUTPUT_TYPE
GPIO_NUMBER

GPIO Private Macros

IS_GPIO_PIN_ACTION

IS_GPIO_PIN

IS_GPIO_PULL

IS_GPIO_SPEED

IS_GPIO_MODE

GPIO pull

GPIO_NOPULL No Pull-up or Pull-down activation

GPIO_PULLUP Pull-up activation

GPIO_PULLDOWN Pull-down activation

GPIO speed

GPIO_SPEED_VERY_LOW Very Low speed

GPIO_SPEED_LOW Low speed

GPIO_SPEED_MEDIUM Medium speed

GPIO_SPEED_HIGH High speed

20 HAL GPIO Extension Driver

20.1 HAL GPIO Extension Driver

20.2 GPIOEx Firmware driver defines

20.2.1 GPIOEx

GPIOEx Alternate function selection

GPIO_AF0_MCO	MCO Alternate Function mapping
GPIO_AF0_TAMPER	TAMPER Alternate Function mapping
GPIO_AF0_SWJ	SWJ (SWD and JTAG) Alternate Function mapping
GPIO_AF0_TRACE	TRACE Alternate Function mapping
GPIO_AF0_RTC_50Hz	RTC_OUT Alternate Function mapping
GPIO_AF1_TIM2	TIM2 Alternate Function mapping
GPIO_AF2_TIM3	TIM3 Alternate Function mapping
GPIO_AF2_TIM4	TIM4 Alternate Function mapping
GPIO_AF2_TIM5	TIM5 Alternate Function mapping
GPIO_AF3_TIM9	TIM9 Alternate Function mapping
GPIO_AF3_TIM10	TIM10 Alternate Function mapping
GPIO_AF3_TIM11	TIM11 Alternate Function mapping
GPIO_AF4_I2C1	I2C1 Alternate Function mapping
GPIO_AF4_I2C2	I2C2 Alternate Function mapping
GPIO_AF5_SPI1	SPI1/I2S1 Alternate Function mapping
GPIO_AF5_SPI2	SPI2/I2S2 Alternate Function mapping
GPIO_AF6_SPI3	SPI3/I2S3 Alternate Function mapping
GPIO_AF7_USART1	USART1 Alternate Function mapping
GPIO_AF7_USART2	USART2 Alternate Function mapping
GPIO_AF7_USART3	USART3 Alternate Function mapping
GPIO_AF8_UART4	UART4 Alternate Function mapping
GPIO_AF8_UART5	UART5 Alternate Function mapping
GPIO_AF11_LCD	LCD Alternate Function mapping
GPIO_AF12_FSMC	FSMC Alternate Function mapping
GPIO_AF12_SDIO	SDIO Alternate Function mapping
GPIO_AF14_TIM_IC1	TIMER INPUT CAPTURE Alternate Function mapping
GPIO_AF14_TIM_IC2	TIMER INPUT CAPTURE Alternate Function mapping
GPIO_AF14_TIM_IC3	TIMER INPUT CAPTURE Alternate Function mapping

GPIO_AF14_TIM_IC4 TIMER INPUT CAPTURE Alternate Function mapping

GPIO_AF15_EVENTOUT EVENTOUT Alternate Function mapping

GPIOEx Private Macros

IS_GPIO_AF

GPIO_GET_INDEX

21 HAL I2C Generic Driver

21.1 HAL I2C Generic Driver

21.2 I2C Firmware driver registers structures

21.2.1 I2C_InitTypeDef

Data Fields

- *uint32_t ClockSpeed*
- *uint32_t DutyCycle*
- *uint32_t OwnAddress1*
- *uint32_t AddressingMode*
- *uint32_t DualAddressMode*
- *uint32_t OwnAddress2*
- *uint32_t GeneralCallMode*
- *uint32_t NoStretchMode*

Field Documentation

- *uint32_t I2C_InitTypeDef::ClockSpeed*
Specifies the clock frequency. This parameter must be set to a value lower than 400kHz
- *uint32_t I2C_InitTypeDef::DutyCycle*
Specifies the I2C fast mode duty cycle. This parameter can be a value of [I2C_duty_cycle_in_fast_mode](#)
- *uint32_t I2C_InitTypeDef::OwnAddress1*
Specifies the first device own address. This parameter can be a 7-bit or 10-bit address.
- *uint32_t I2C_InitTypeDef::AddressingMode*
Specifies if 7-bit or 10-bit addressing mode is selected. This parameter can be a value of [I2C_addressing_mode](#)
- *uint32_t I2C_InitTypeDef::DualAddressMode*
Specifies if dual addressing mode is selected. This parameter can be a value of [I2C_dual_addressing_mode](#)
- *uint32_t I2C_InitTypeDef::OwnAddress2*
Specifies the second device own address if dual addressing mode is selected. This parameter can be a 7-bit address.
- *uint32_t I2C_InitTypeDef::GeneralCallMode*
Specifies if general call mode is selected. This parameter can be a value of [I2C_general_call_addressing_mode](#)
- *uint32_t I2C_InitTypeDef::NoStretchMode*
Specifies if nostretch mode is selected. This parameter can be a value of [I2C_nostretch_mode](#)

21.2.2 I2C_HandleTypeDef

Data Fields

- *I2C_TypeDef * Instance*
- *I2C_InitTypeDef Init*
- *uint8_t * pBuffPtr*
- *uint16_t XferSize*
- *__IO uint16_t XferCount*
- *DMA_HandleTypeDef * hdmatx*
- *DMA_HandleTypeDef * hdmarx*
- *HAL_LockTypeDef Lock*
- *__IO HAL_I2C_StateTypeDef State*
- *__IO uint32_t ErrorCode*

Field Documentation

- *I2C_TypeDef* I2C_HandleTypeDef::Instance*
I2C registers base address
- *I2C_InitTypeDef I2C_HandleTypeDef::Init*
I2C communication parameters
- *uint8_t* I2C_HandleTypeDef::pBuffPtr*
Pointer to I2C transfer buffer
- *uint16_t I2C_HandleTypeDef::XferSize*
I2C transfer size
- *__IO uint16_t I2C_HandleTypeDef::XferCount*
I2C transfer counter
- *DMA_HandleTypeDef* I2C_HandleTypeDef::hdmatx*
I2C Tx DMA handle parameters
- *DMA_HandleTypeDef* I2C_HandleTypeDef::hdmarx*
I2C Rx DMA handle parameters
- *HAL_LockTypeDef I2C_HandleTypeDef::Lock*
I2C locking object
- *__IO HAL_I2C_StateTypeDef I2C_HandleTypeDef::State*
I2C communication state
- *__IO uint32_t I2C_HandleTypeDef::ErrorCode*

21.3 I2C Firmware driver API description

21.3.1 How to use this driver

The I2C HAL driver can be used as follows:

1. Declare a I2C_HandleTypeDef handle structure, for example: I2C_HandleTypeDef hi2c;
2. Initialize the I2C low level resources by implement the HAL_I2C_MspInit() API:
 - a. Enable the I2Cx interface clock
 - b. I2C pins configuration
 - Enable the clock for the I2C GPIOs
 - Configure I2C pins as alternate function open-drain
 - c. NVIC configuration if you need to use interrupt process
 - Configure the I2Cx interrupt priority
 - Enable the NVIC I2C IRQ Channel
 - d. DMA Configuration if you need to use DMA process

- Declare a DMA_HandleTypeDef handle structure for the transmit or receive channel
 - Enable the DMAx interface clock using
 - Configure the DMA handle parameters
 - Configure the DMA Tx or Rx Channel
 - Associate the initialized DMA handle to the hi2c DMA Tx or Rx handle
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Channel
3. Configure the Communication Speed, Duty cycle, Addressing mode, Own Address1, Dual Addressing mode, Own Address2, General call and Nostretch mode in the hi2c Init structure.
 4. Initialize the I2C registers by calling the HAL_I2C_Init(), configures also the low level Hardware (GPIO, CLOCK, NVIC...etc) by calling the customized HAL_I2C_MspInit(&hi2c) API.
 5. To check if target device is ready for communication, use the function HAL_I2C_IsDeviceReady()
 6. For I2C IO and IO MEM operations, three operation modes are available within this driver :

Polling mode IO operation

- Transmit in master mode an amount of data in blocking mode using HAL_I2C_Master_Transmit()
- Receive in master mode an amount of data in blocking mode using HAL_I2C_Master_Receive()
- Transmit in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Transmit()
- Receive in slave mode an amount of data in blocking mode using HAL_I2C_Slave_Receive()

Polling mode IO MEM operation

- Write an amount of data in blocking mode to a specific memory address using HAL_I2C_Mem_Write()
- Read an amount of data in blocking mode from a specific memory address using HAL_I2C_Mem_Read()

Interrupt mode IO operation

- Transmit in master mode an amount of data in non blocking mode using HAL_I2C_Master_Transmit_IT()
- At transmission end of transfer HAL_I2C_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode using HAL_I2C_Master_Receive_IT()
- At reception end of transfer HAL_I2C_MasterRxTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxTxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode using HAL_I2C_Slave_Transmit_IT()

- At transmission end of transfer HAL_I2C_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode using HAL_I2C_Slave_Receive_IT()
- At reception end of transfer HAL_I2C_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

Interrupt mode IO MEM operation

- Write an amount of data in no-blocking mode with Interrupt to a specific memory address using HAL_I2C_Mem_Write_IT()
- At MEM end of write transfer HAL_I2C_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback
- Read an amount of data in no-blocking mode with Interrupt from a specific memory address using HAL_I2C_Mem_Read_IT()
- At MEM end of read transfer HAL_I2C_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

DMA mode IO operation

- Transmit in master mode an amount of data in non blocking mode (DMA) using HAL_I2C_Master_Transmit_DMA()
- At transmission end of transfer HAL_I2C_MasterTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterTxCpltCallback
- Receive in master mode an amount of data in non blocking mode (DMA) using HAL_I2C_Master_Receive_DMA()
- At reception end of transfer HAL_I2C_MasterRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MasterRxCpltCallback
- Transmit in slave mode an amount of data in non blocking mode (DMA) using HAL_I2C_Slave_Transmit_DMA()
- At transmission end of transfer HAL_I2C_SlaveTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveTxCpltCallback
- Receive in slave mode an amount of data in non blocking mode (DMA) using HAL_I2C_Slave_Receive_DMA()
- At reception end of transfer HAL_I2C_SlaveRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_SlaveRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

DMA mode IO MEM operation

- Write an amount of data in no-blocking mode with DMA to a specific memory address using HAL_I2C_Mem_Write_DMA()

- At MEM end of write transfer HAL_I2C_MemTxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemTxCpltCallback
- Read an amount of data in no-blocking mode with DMA from a specific memory address using HAL_I2C_Mem_Read_DMA()
- At MEM end of read transfer HAL_I2C_MemRxCpltCallback is executed and user can add his own code by customization of function pointer HAL_I2C_MemRxCpltCallback
- In case of transfer Error, HAL_I2C_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_I2C_ErrorCallback

I2C HAL driver macros list

Below the list of most used macros in I2C HAL driver.

- `__HAL_I2C_ENABLE`: Enable the I2C peripheral
- `__HAL_I2C_DISABLE`: Disable the I2C peripheral
- `__HAL_I2C_GET_FLAG`: Checks whether the specified I2C flag is set or not
- `__HAL_I2C_CLEAR_FLAG`: Clear the specified I2C pending flag
- `__HAL_I2C_ENABLE_IT`: Enable the specified I2C interrupt
- `__HAL_I2C_DISABLE_IT`: Disable the specified I2C interrupt



You can refer to the I2C HAL driver header file for more useful macros

21.3.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Cx peripheral:

- User must Implement HAL_I2C_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_I2C_Init() to configure the selected device with the selected configuration:
 - Communication Speed
 - Duty cycle
 - Addressing mode
 - Own Address 1
 - Dual Addressing mode
 - Own Address 2
 - General call mode
 - Nostretch mode
- Call the function HAL_I2C_DeInit() to restore the default configuration of the selected I2Cx peripheral.

This section contains the following APIs:

- [*HAL_I2C_Init\(\)*](#)
- [*HAL_I2C_DeInit\(\)*](#)
- [*HAL_I2C_MspInit\(\)*](#)
- [*HAL_I2C_MspDeInit\(\)*](#)

21.3.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2C data transfers.

1. There are two modes of transfer:

- Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2C IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
 - HAL_I2C_Master_Transmit()
 - HAL_I2C_Master_Receive()
 - HAL_I2C_Slave_Transmit()
 - HAL_I2C_Slave_Receive()
 - HAL_I2C_Mem_Write()
 - HAL_I2C_Mem_Read()
 - HAL_I2C_IsDeviceReady()
 3. No-Blocking mode functions with Interrupt are :
 - HAL_I2C_Master_Transmit_IT()
 - HAL_I2C_Master_Receive_IT()
 - HAL_I2C_Slave_Transmit_IT()
 - HAL_I2C_Slave_Receive_IT()
 - HAL_I2C_Mem_Write_IT()
 - HAL_I2C_Mem_Read_IT()
 4. No-Blocking mode functions with DMA are :
 - HAL_I2C_Master_Transmit_DMA()
 - HAL_I2C_Master_Receive_DMA()
 - HAL_I2C_Slave_Transmit_DMA()
 - HAL_I2C_Slave_Receive_DMA()
 - HAL_I2C_Mem_Write_DMA()
 - HAL_I2C_Mem_Read_DMA()
 5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - HAL_I2C_MemTxCpltCallback()
 - HAL_I2C_MemRxCpltCallback()
 - HAL_I2C_MasterTxCpltCallback()
 - HAL_I2C_MasterRxCpltCallback()
 - HAL_I2C_SlaveTxCpltCallback()
 - HAL_I2C_SlaveRxCpltCallback()
 - HAL_I2C_ErrorCallback()

This section contains the following APIs:

- [*HAL_I2C_Master_Transmit\(\)*](#)
- [*HAL_I2C_Master_Receive\(\)*](#)
- [*HAL_I2C_Slave_Transmit\(\)*](#)
- [*HAL_I2C_Slave_Receive\(\)*](#)
- [*HAL_I2C_Master_Transmit_IT\(\)*](#)
- [*HAL_I2C_Master_Receive_IT\(\)*](#)
- [*HAL_I2C_Slave_Transmit_IT\(\)*](#)
- [*HAL_I2C_Slave_Receive_IT\(\)*](#)
- [*HAL_I2C_Master_Transmit_DMA\(\)*](#)
- [*HAL_I2C_Master_Receive_DMA\(\)*](#)
- [*HAL_I2C_Slave_Transmit_DMA\(\)*](#)
- [*HAL_I2C_Slave_Receive_DMA\(\)*](#)
- [*HAL_I2C_Mem_Write\(\)*](#)
- [*HAL_I2C_Mem_Read\(\)*](#)

- [HAL_I2C_Mem_Write_IT\(\)](#)
- [HAL_I2C_Mem_Read_IT\(\)](#)
- [HAL_I2C_Mem_Write_DMA\(\)](#)
- [HAL_I2C_Mem_Read_DMA\(\)](#)
- [HAL_I2C_IsDeviceReady\(\)](#)
- [HAL_I2C_EV_IRQHandler\(\)](#)
- [HAL_I2C_ER_IRQHandler\(\)](#)
- [HAL_I2C_MasterTxCpltCallback\(\)](#)
- [HAL_I2C_MasterRxCpltCallback\(\)](#)
- [HAL_I2C_SlaveTxCpltCallback\(\)](#)
- [HAL_I2C_SlaveRxCpltCallback\(\)](#)
- [HAL_I2C_MemTxCpltCallback\(\)](#)
- [HAL_I2C_MemRxCpltCallback\(\)](#)
- [HAL_I2C_ErrorCallback\(\)](#)

21.3.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- [HAL_I2C_GetState\(\)](#)
- [HAL_I2C_GetError\(\)](#)

21.3.5 HAL_I2C_Init

Function Name	HAL_StatusTypeDef HAL_I2C_Init (I2C_HandleTypeDef * hi2c)
Function Description	Initializes the I2C according to the specified parameters in the I2C_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none"> • HAL status

21.3.6 HAL_I2C_DeInit

Function Name	HAL_StatusTypeDef HAL_I2C_DeInit (I2C_HandleTypeDef * hi2c)
Function Description	DeInitializes the I2C peripheral.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none"> • HAL status

21.3.7 HAL_I2C_MspInit

Function Name	void HAL_I2C_MspInit (I2C_HandleTypeDef * hi2c)
Function Description	I2C MSP Init.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none"> • None

21.3.8 HAL_I2C_MspDeInit



Function Name	void HAL_I2C_MspDeInit (I2C_HandleTypeDef * hi2c)
Function Description	I2C MSP DeInit.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none"> • None

21.3.9 HAL_I2C_Master_Transmit

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Transmit (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits in master mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

21.3.10 HAL_I2C_Master_Receive

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Receive (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receives in master mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

21.3.11 HAL_I2C_Slave_Transmit

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Transmit (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmits in slave mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

21.3.12 HAL_I2C_Slave_Receive

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Receive (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive in slave mode an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

21.3.13 HAL_I2C_Master_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Transmit_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)
Function Description	Transmit in master mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

21.3.14 HAL_I2C_Master_Receive_IT

Function Name	HAL_StatusTypeDef HAL_I2C_Master_Receive_IT (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)
Function Description	Receive in master mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • DevAddress: Target device address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

21.3.15 HAL_I2C_Slave_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Transmit_IT (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)
Function Description	Transmit in slave mode an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module

- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- HAL status

21.3.16 HAL_I2C_Slave_Receive_IT

Function Name HAL_StatusTypeDef HAL_I2C_Slave_Receive_IT
(I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function Description Receive in slave mode an amount of data in no-blocking mode with Interrupt.

Parameters

- **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- HAL status

21.3.17 HAL_I2C_Master_Transmit_DMA

Function Name HAL_StatusTypeDef HAL_I2C_Master_Transmit_DMA
(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function Description Transmit in master mode an amount of data in no-blocking mode with DMA.

Parameters

- **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- HAL status

21.3.18 HAL_I2C_Master_Receive_DMA

Function Name HAL_StatusTypeDef HAL_I2C_Master_Receive_DMA
(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size)

Function Description Receive in master mode an amount of data in no-blocking mode with DMA.

Parameters

- **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
- **DevAddress:** Target device address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- HAL status

21.3.19 HAL_I2C_Slave_Transmit_DMA

Function Name HAL_StatusTypeDef HAL_I2C_Slave_Transmit_DMA
(I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)

Function Description	Transmit in slave mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

21.3.20 HAL_I2C_Slave_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_I2C_Slave_Receive_DMA (I2C_HandleTypeDef * hi2c, uint8_t * pData, uint16_t Size)
Function Description	Receive in slave mode an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

21.3.21 HAL_I2C_Mem_Write

Function Name	HAL_StatusTypeDef HAL_I2C_Mem_Write (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Write an amount of data in blocking mode to a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • DevAddress: Target device address • MemAddress: Internal memory address • MemAddSize: Size of internal memory address • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

21.3.22 HAL_I2C_Mem_Read

Function Name	HAL_StatusTypeDef HAL_I2C_Mem_Read (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Read an amount of data in blocking mode from a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • DevAddress: Target device address

- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent
- **Timeout:** Timeout duration

Return values

- HAL status

21.3.23 HAL_I2C_Mem_Write_IT

Function Name

HAL_StatusTypeDef HAL_I2C_Mem_Write_IT
(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function Description

Write an amount of data in no-blocking mode with Interrupt to a specific memory address.

Parameters

- **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- HAL status

21.3.24 HAL_I2C_Mem_Read_IT

Function Name

HAL_StatusTypeDef HAL_I2C_Mem_Read_IT
(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function Description

Read an amount of data in no-blocking mode with Interrupt from a specific memory address.

Parameters

- **hi2c:** pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
- **DevAddress:** Target device address
- **MemAddress:** Internal memory address
- **MemAddSize:** Size of internal memory address
- **pData:** Pointer to data buffer
- **Size:** Amount of data to be sent

Return values

- HAL status

21.3.25 HAL_I2C_Mem_Write_DMA

Function Name

HAL_StatusTypeDef HAL_I2C_Mem_Write_DMA
(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)

Function Description

Write an amount of data in no-blocking mode with DMA to a specific memory address.

Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • DevAddress: Target device address • MemAddress: Internal memory address • MemAddSize: Size of internal memory address • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

21.3.26 HAL_I2C_Mem_Read_DMA

Function Name	HAL_StatusTypeDef HAL_I2C_Mem_Read_DMA (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size)
Function Description	Reads an amount of data in no-blocking mode with DMA from a specific memory address.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • DevAddress: Target device address • MemAddress: Internal memory address • MemAddSize: Size of internal memory address • pData: Pointer to data buffer • Size: Amount of data to be read
Return values	<ul style="list-style-type: none"> • HAL status

21.3.27 HAL_I2C_IsDeviceReady

Function Name	HAL_StatusTypeDef HAL_I2C_IsDeviceReady (I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)
Function Description	Checks if target device is ready for communication.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module • DevAddress: Target device address • Trials: Number of trials • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This function is used with Memory devices

21.3.28 HAL_I2C_EV_IRQHandler

Function Name	void HAL_I2C_EV_IRQHandler (I2C_HandleTypeDef * hi2c)
Function Description	This function handles I2C event interrupt request.
Parameters	<ul style="list-style-type: none"> • hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none"> • HAL status

21.3.29 HAL_I2C_ER_IRQHandler

Function Name	void HAL_I2C_ER_IRQHandler (I2C_HandleTypeDef * hi2c)
Function Description	This function handles I2C error interrupt request.
Parameters	<ul style="list-style-type: none">• hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none">• HAL status

21.3.30 HAL_I2C_MasterTxCpltCallback

Function Name	void HAL_I2C_MasterTxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Master Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none">• None

21.3.31 HAL_I2C_MasterRxCpltCallback

Function Name	void HAL_I2C_MasterRxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Master Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none">• None

21.3.32 HAL_I2C_SlaveTxCpltCallback

Function Name	void HAL_I2C_SlaveTxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Slave Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none">• None

21.3.33 HAL_I2C_SlaveRxCpltCallback

Function Name	void HAL_I2C_SlaveRxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Slave Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none">• None

21.3.34 HAL_I2C_MemTxCpltCallback

Function Name	void HAL_I2C_MemTxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Memory Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none"> None

21.3.35 HAL_I2C_MemRxCpltCallback

Function Name	void HAL_I2C_MemRxCpltCallback (I2C_HandleTypeDef * hi2c)
Function Description	Memory Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none"> None

21.3.36 HAL_I2C_ErrorCallback

Function Name	void HAL_I2C_ErrorCallback (I2C_HandleTypeDef * hi2c)
Function Description	I2C error callbacks.
Parameters	<ul style="list-style-type: none"> hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none"> None

21.3.37 HAL_I2C_GetState

Function Name	HAL_I2C_StateTypeDef HAL_I2C_GetState (I2C_HandleTypeDef * hi2c)
Function Description	Returns the I2C state.
Parameters	<ul style="list-style-type: none"> hi2c: pointer to a I2C_HandleTypeDef structure that contains the configuration information for I2C module
Return values	<ul style="list-style-type: none"> HAL state

21.3.38 HAL_I2C_GetError

Function Name	uint32_t HAL_I2C_GetError (I2C_HandleTypeDef * hi2c)
Function Description	Return the I2C error code.
Parameters	<ul style="list-style-type: none"> hi2c: : pointer to a I2C_HandleTypeDef structure that contains the configuration information for the specified I2C.
Return values	<ul style="list-style-type: none"> I2C Error Code

21.4 I2C Firmware driver defines

21.4.1 I2C

I2C_addressing_mode

I2C_ADDRESSINGMODE_7BIT

I2C_ADDRESSINGMODE_10BIT

IS_I2C_ADDRESSING_MODE

I2C_Clock_Speed_definition

IS_I2C_CLOCK_SPEED

I2C_dual_addressing_mode

I2C_DUALADDRESS_DISABLE

I2C_DUALADDRESS_ENABLE

IS_I2C_DUAL_ADDRESS

I2C_duty_cycle_in_fast_mode

I2C_DUTYCYCLE_2

I2C_DUTYCYCLE_16_9

IS_I2C_DUTY_CYCLE

I2C Error Codes

HAL_I2C_ERROR_NONE No error

HAL_I2C_ERROR_BERR BERR error

HAL_I2C_ERROR_ARLO ARLO error

HAL_I2C_ERROR_AF AF error

HAL_I2C_ERROR_OVR OVR error

HAL_I2C_ERROR_DMA DMA transfer error

HAL_I2C_ERROR_TIMEOUT Timeout error

I2C Exported Macros**__HAL_I2C_RESET_HANDLE_STATE** **Description:**

- Reset I2C handle state.

Parameters:

- __HANDLE__**: specifies the I2C Handle.
This parameter can be I2C where x: 1, 2, or 3 to select the I2C peripheral.

Return value:

- None

__HAL_I2C_ENABLE_IT**Description:**

- Enable or disable the specified I2C interrupts.

Parameters:

- __HANDLE__**: specifies the I2C Handle.
This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.
- __INTERRUPT__**: specifies the interrupt source to enable or disable. This parameter

can be one of the following values:

- I2C_IT_BUF: Buffer interrupt enable
- I2C_IT_EVT: Event interrupt enable
- I2C_IT_ERR: Error interrupt enable

Return value:

- None

`__HAL_I2C_DISABLE_IT`

`__HAL_I2C_GET_IT_SOURCE`

Description:

- Checks if the specified I2C interrupt source is enabled or disabled.

Parameters:

- `__HANDLE__`: specifies the I2C Handle.
This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.
- `__INTERRUPT__`: specifies the I2C interrupt source to check. This parameter can be one of the following values:
 - I2C_IT_BUF: Buffer interrupt enable
 - I2C_IT_EVT: Event interrupt enable
 - I2C_IT_ERR: Error interrupt enable

Return value:

- The: new state of `__INTERRUPT__` (TRUE or FALSE).

`__HAL_I2C_GET_FLAG`

Description:

- Checks whether the specified I2C flag is set or not.

Parameters:

- `__HANDLE__`: specifies the I2C Handle.
This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - I2C_FLAG_OVR: Overrun/Underrun flag
 - I2C_FLAG_AF: Acknowledge failure flag
 - I2C_FLAG_ARLO: Arbitration lost flag
 - I2C_FLAG_BERR: Bus error flag
 - I2C_FLAG_TXE: Data register empty flag
 - I2C_FLAG_RXNE: Data register not empty flag
 - I2C_FLAG_STOPF: Stop detection flag
 - I2C_FLAG_ADD10: 10-bit header sent flag
 - I2C_FLAG_BTF: Byte transfer finished

- flag
- I2C_FLAG_ADDR: Address sent flag
Address matched flag
- I2C_FLAG_SB: Start bit flag
- I2C_FLAG_DUALF: Dual flag
- I2C_FLAG_GENCALL: General call
header flag
- I2C_FLAG_TRA: Transmitter/Receiver
flag
- I2C_FLAG_BUSY: Bus busy flag
- I2C_FLAG_MSL: Master/Slave flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

Description:

- Clears the I2C pending flags which are cleared by writing 0 in a specific bit.

Parameters:

- __HANDLE__: specifies the I2C Handle.
This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.
- __FLAG__: specifies the flag to clear. This parameter can be any combination of the following values:
 - I2C_FLAG_OVR: Overrun/Underrun flag (Slave mode)
 - I2C_FLAG_AF: Acknowledge failure flag
 - I2C_FLAG_ARLO: Arbitration lost flag (Master mode)
 - I2C_FLAG_BERR: Bus error flag

Return value:

- None

Description:

- Clears the I2C ADDR pending flag.

Parameters:

- __HANDLE__: specifies the I2C Handle.
This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.

Return value:

- None

Description:

- Clears the I2C STOPF pending flag.

Parameters:

- __HANDLE__: specifies the I2C Handle.

`__HAL_I2C_CLEAR_FLAG`

`__HAL_I2C_CLEAR_ADDRFLAG`

`__HAL_I2C_CLEAR_STOPFLAG`

This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.

Return value:

- None

Description:

- Enable the I2C peripheral.

Parameters:

- `__HANDLE__`: specifies the I2C Handle.
This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.

Return value:

- None

Description:

- Disable the I2C peripheral.

Parameters:

- `__HANDLE__`: specifies the I2C Handle.
This parameter can be I2Cx where x: 1 or 2 to select the I2C peripheral.

Return value:

- None

`__HAL_I2C_ENABLE`

`__HAL_I2C_DISABLE`

I2C_Flag_definition

`I2C_FLAG_OVR`

`I2C_FLAG_AF`

`I2C_FLAG_ARLO`

`I2C_FLAG_BERR`

`I2C_FLAG_TXE`

`I2C_FLAG_RXNE`

`I2C_FLAG_STOPF`

`I2C_FLAG_ADD10`

`I2C_FLAG_BTF`

`I2C_FLAG_ADDR`

`I2C_FLAG_SB`

`I2C_FLAG_DUALF`

`I2C_FLAG_GENCALL`

`I2C_FLAG_TRA`

`I2C_FLAG_BUSY`

`I2C_FLAG_MSL`

`I2C_FLAG_MASK`

I2C_general_call_addressing_mode

I2C_GENERALCALL_DISABLE

I2C_GENERALCALL_ENABLE

IS_I2C_GENERAL_CALL

I2C_Interrupt_configuration_definition

I2C_IT_BUF

I2C_IT_EVT

I2C_IT_ERR

I2C_Memory_Address_Size

I2C_MEMADD_SIZE_8BIT

I2C_MEMADD_SIZE_16BIT

IS_I2C_MEMADD_SIZE

I2C_nostretch_mode

I2C_NOSTRETCH_DISABLE

I2C_NOSTRETCH_ENABLE

IS_I2C_NO_STRETCH

I2C_Own_Address1_definition

IS_I2C_OWN_ADDRESS1

I2C_Own_Address2_definition

IS_I2C_OWN_ADDRESS2

I2C Private Constants

I2C_TIMEOUT_FLAG

I2C_TIMEOUT_ADDR_SLAVE

I2C_MIN_PCLK_FREQ

I2C Private Macros

I2C_FREQ_RANGE

I2C_RISE_TIME

I2C_SPEED_STANDARD

I2C_SPEED_FAST

I2C_SPEED

I2C_7BIT_ADD_WRITE

I2C_7BIT_ADD_READ

I2C_10BIT_ADDRESS

I2C_10BIT_HEADER_WRITE

I2C_10BIT_HEADER_READ

I2C_MEM_ADD_MSB

I2C_MEM_ADD_LSB

22 HAL I2S Generic Driver

22.1 HAL I2S Generic Driver

22.2 I2S Firmware driver registers structures

22.2.1 I2S_InitTypeDef

Data Fields

- *uint32_t Mode*
- *uint32_t Standard*
- *uint32_t DataFormat*
- *uint32_t MCLKOutput*
- *uint32_t AudioFreq*
- *uint32_t CPOL*

Field Documentation

- *uint32_t I2S_InitTypeDef::Mode*
Specifies the I2S operating mode. This parameter can be a value of [I2S_Mode](#)
- *uint32_t I2S_InitTypeDef::Standard*
Specifies the standard used for the I2S communication. This parameter can be a value of [I2S_Standard](#)
- *uint32_t I2S_InitTypeDef::DataFormat*
Specifies the data format for the I2S communication. This parameter can be a value of [I2S_Data_Format](#)
- *uint32_t I2S_InitTypeDef::MCLKOutput*
Specifies whether the I2S MCLK output is enabled or not. This parameter can be a value of [I2S_MCLK_Output](#)
- *uint32_t I2S_InitTypeDef::AudioFreq*
Specifies the frequency selected for the I2S communication. This parameter can be a value of [I2S_Audio_Frequency](#)
- *uint32_t I2S_InitTypeDef::CPOL*
Specifies the idle state of the I2S clock. This parameter can be a value of [I2S_Clock_Polarity](#)

22.2.2 I2S_HandleTypeDef

Data Fields

- *SPI_TypeDef * Instance*
- *I2S_InitTypeDef Init*
- *uint16_t * pTxBuffPtr*
- *__IO uint16_t TxXferSize*
- *__IO uint16_t TxXferCount*
- *uint16_t * pRxBuffPtr*
- *__IO uint16_t RxXferSize*

- `__IO uint16_t RxXferCount`
- `DMA_HandleTypeDef * hdmatx`
- `DMA_HandleTypeDef * hdmarx`
- `__IO HAL_LockTypeDef Lock`
- `__IO HAL_I2S_StateTypeDef State`
- `__IO uint32_t ErrorCode`

Field Documentation

- `SPI_TypeDef* I2S_HandleTypeDef::Instance`
- `I2S_InitTypeDef I2S_HandleTypeDef::Init`
- `uint16_t* I2S_HandleTypeDef::pTxBuffPtr`
- `__IO uint16_t I2S_HandleTypeDef::TxXferSize`
- `__IO uint16_t I2S_HandleTypeDef::TxXferCount`
- `uint16_t* I2S_HandleTypeDef::pRxBuffPtr`
- `__IO uint16_t I2S_HandleTypeDef::RxXferSize`
- `__IO uint16_t I2S_HandleTypeDef::RxXferCount`
- `DMA_HandleTypeDef* I2S_HandleTypeDef::hdmatx`
- `DMA_HandleTypeDef* I2S_HandleTypeDef::hdmarx`
- `__IO HAL_LockTypeDef I2S_HandleTypeDef::Lock`
- `__IO HAL_I2S_StateTypeDef I2S_HandleTypeDef::State`
- `__IO uint32_t I2S_HandleTypeDef::ErrorCode`

22.3 I2S Firmware driver API description

22.3.1 How to use this driver

The I2S HAL driver can be used as follow:

1. Declare a `I2S_HandleTypeDef` handle structure.
2. Initialize the I2S low level resources by implement the `HAL_I2S_MspInit()` API:
 - a. Enable the SPIx interface clock.
 - b. I2S pins configuration:
 - Enable the clock for the I2S GPIOs.
 - Configure these I2S pins as alternate function.
 - c. NVIC configuration if you need to use interrupt process (`HAL_I2S_Transmit_IT()` and `HAL_I2S_Receive_IT()` APIs).
 - Configure the I2Sx interrupt priority.
 - Enable the NVIC I2S IRQ handle.
 - d. DMA Configuration if you need to use DMA process (`HAL_I2S_Transmit_DMA()` and `HAL_I2S_Receive_DMA()` APIs):
 - Declare a DMA handle structure for the Tx/Rx Channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx Channel.
 - Associate the initialized DMA handle to the I2S DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx Channel.
3. Program the Mode, Standard, Data Format, MCLK Output, Audio frequency and Polarity using `HAL_I2S_Init()` function. The specific I2S interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the

macros `__HAL_I2S_ENABLE_IT()` and `__HAL_I2S_DISABLE_IT()` inside the transmit and receive process. Make sure that either: External clock source is configured after setting correctly the define constant `HSE_VALUE` in the `stm32l1xx_hal_conf.h` file.

4. Three mode of operations are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using `HAL_I2S_Transmit()`
- Receive an amount of data in blocking mode using `HAL_I2S_Receive()`

Interrupt mode IO operation

- Send an amount of data in non blocking mode using `HAL_I2S_Transmit_IT()`
- At transmission end of half transfer `HAL_I2S_TxHalfCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_TxHalfCpltCallback`
- At transmission end of transfer `HAL_I2S_TxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_TxCpltCallback`
- Receive an amount of data in non blocking mode using `HAL_I2S_Receive_IT()`
- At reception end of half transfer `HAL_I2S_RxHalfCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_RxHalfCpltCallback`
- At reception end of transfer `HAL_I2S_RxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_RxCpltCallback`
- In case of transfer Error, `HAL_I2S_ErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_I2S_ErrorCallback`

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using `HAL_I2S_Transmit_DMA()`
- At transmission end of half transfer `HAL_I2S_TxHalfCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_TxHalfCpltCallback`
- At transmission end of transfer `HAL_I2S_TxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_TxCpltCallback`
- Receive an amount of data in non blocking mode (DMA) using `HAL_I2S_Receive_DMA()`
- At reception end of half transfer `HAL_I2S_RxHalfCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_RxHalfCpltCallback`
- At reception end of transfer `HAL_I2S_RxCpltCallback` is executed and user can add his own code by customization of function pointer `HAL_I2S_RxCpltCallback`
- In case of transfer Error, `HAL_I2S_ErrorCallback()` function is executed and user can add his own code by customization of function pointer `HAL_I2S_ErrorCallback`
- Pause the DMA Transfer using `HAL_I2S_DMAPause()`
- Resume the DMA Transfer using `HAL_I2S_DMAResume()`
- Stop the DMA Transfer using `HAL_I2S_DMAStop()`

I2S HAL driver macros list

Below the list of most used macros in USART HAL driver.

- `__HAL_I2S_ENABLE`: Enable the specified SPI peripheral (in I2S mode)
- `__HAL_I2S_DISABLE`: Disable the specified SPI peripheral (in I2S mode)
- `__HAL_I2S_ENABLE_IT` : Enable the specified I2S interrupts
- `__HAL_I2S_DISABLE_IT` : Disable the specified I2S interrupts
- `__HAL_I2S_GET_FLAG`: Check whether the specified I2S flag is set or not



You can refer to the I2S HAL driver header file for more useful macros

22.3.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the I2Sx peripheral in simplex mode:

- User must Implement `HAL_I2S_MspInit()` function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function `HAL_I2S_Init()` to configure the selected device with the selected configuration:
 - Mode
 - Standard
 - Data Format
 - MCLK Output
 - Audio frequency
 - Polarity
- Call the function `HAL_I2S_DeInit()` to restore the default configuration of the selected I2Sx peripheral.

This section contains the following APIs:

- [*HAL_I2S_Init\(\)*](#)
- [*HAL_I2S_DeInit\(\)*](#)
- [*HAL_I2S_MspInit\(\)*](#)
- [*HAL_I2S_MspDeInit\(\)*](#)

22.3.3 IO operation functions

This subsection provides a set of functions allowing to manage the I2S data transfers.

1. There are two modes of transfer:
 - Blocking mode : The communication is performed in the polling mode. The status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode : The communication is performed using Interrupts or DMA. These functions return the status of the transfer startup. The end of the data processing will be indicated through the dedicated I2S IRQ when using Interrupt mode or the DMA IRQ when using DMA mode.
2. Blocking mode functions are :
 - `HAL_I2S_Transmit()`
 - `HAL_I2S_Receive()`
3. No-Blocking mode functions with Interrupt are :
 - `HAL_I2S_Transmit_IT()`

- HAL_I2S_Receive_IT()
- 4. No-Blocking mode functions with DMA are :
 - HAL_I2S_Transmit_DMA()
 - HAL_I2S_Receive_DMA()
- 5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - HAL_I2S_TxCpltCallback()
 - HAL_I2S_RxCpltCallback()
 - HAL_I2S_ErrorCallback()

This section contains the following APIs:

- [HAL_I2S_Transmit\(\)](#)
- [HAL_I2S_Receive\(\)](#)
- [HAL_I2S_Transmit_IT\(\)](#)
- [HAL_I2S_Receive_IT\(\)](#)
- [HAL_I2S_Transmit_DMA\(\)](#)
- [HAL_I2S_Receive_DMA\(\)](#)
- [HAL_I2S_DMAPause\(\)](#)
- [HAL_I2S_DMAResume\(\)](#)
- [HAL_I2S_DMAStop\(\)](#)
- [HAL_I2S_IRQHandler\(\)](#)
- [HAL_I2S_TxHalfCpltCallback\(\)](#)
- [HAL_I2S_TxCpltCallback\(\)](#)
- [HAL_I2S_RxHalfCpltCallback\(\)](#)
- [HAL_I2S_RxCpltCallback\(\)](#)
- [HAL_I2S_ErrorCallback\(\)](#)

22.3.4 Peripheral State and Errors functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- [HAL_I2S_GetState\(\)](#)
- [HAL_I2S_GetError\(\)](#)

22.3.5 HAL_I2S_Init

Function Name	HAL_StatusTypeDef HAL_I2S_Init (I2S_HandleTypeDef * hi2s)
Function Description	Initializes the I2S according to the specified parameters in the I2S_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL status

22.3.6 HAL_I2S_DeInit

Function Name	HAL_StatusTypeDef HAL_I2S_DeInit (I2S_HandleTypeDef * hi2s)
Function Description	DeInitializes the I2S peripheral.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> • HAL status

22.3.7 HAL_I2S_MspInit

Function Name	void HAL_I2S_MspInit (I2S_HandleTypeDef * hi2s)
Function Description	I2S MSP Init.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> None

22.3.8 HAL_I2S_MspDeInit

Function Name	void HAL_I2S_MspDeInit (I2S_HandleTypeDef * hi2s)
Function Description	I2S MSP DeInit.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> None

22.3.9 HAL_I2S_Transmit

Function Name	HAL_StatusTypeDef HAL_I2S_Transmit (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Transmit an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module pData: a 16-bit pointer to data buffer. Size: number of data sample to be sent: Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

22.3.10 HAL_I2S_Receive

Function Name	HAL_StatusTypeDef HAL_I2S_Receive (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module pData: a 16-bit pointer to data buffer. Size: number of data sample to be sent: Timeout: Timeout duration

Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming). • In I2S Master Receiver mode, just after enabling the peripheral the clock will be generate in continouse way and as the I2S is not disabled at the end of the I2S transaction.

22.3.11 HAL_I2S_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_I2S_Transmit_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to data buffer. • Size: number of data sample to be sent:
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

22.3.12 HAL_I2S_Receive_IT

Function Name	HAL_StatusTypeDef HAL_I2S_Receive_IT (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Receive an amount of data in non-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Receive data buffer. • Size: number of data sample to be sent:
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length.

- The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).
- It is recommended to use DMA for the I2S receiver to avoid de-synchronisation between Master and Slave otherwise the I2S interrupt should be optimized.

22.3.13 HAL_I2S_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_I2S_Transmit_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Transmit data buffer. • Size: number of data sample to be sent:
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

22.3.14 HAL_I2S_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_I2S_Receive_DMA (I2S_HandleTypeDef * hi2s, uint16_t * pData, uint16_t Size)
Function Description	Receive an amount of data in non-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module • pData: a 16-bit pointer to the Receive data buffer. • Size: number of data sample to be sent:
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When a 16-bit data frame or a 16-bit data frame extended is selected during the I2S configuration phase, the Size parameter means the number of 16-bit data length in the transaction and when a 24-bit data frame or a 32-bit data frame is selected the Size parameter means the number of 16-bit data length. • The I2S is kept enabled at the end of transaction to avoid the clock de-synchronization between Master and Slave(example: audio streaming).

22.3.15 HAL_I2S_DMAPause

Function Name	HAL_StatusTypeDef HAL_I2S_DMAPause
---------------	---

(I2S_HandleTypeDef * hi2s)

Function Description	Pauses the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> HAL status

22.3.16 HAL_I2S_DMAResume

Function Name	HAL_StatusTypeDef HAL_I2S_DMAResume (I2S_HandleTypeDef * hi2s)
Function Description	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> HAL status

22.3.17 HAL_I2S_DMAStop

Function Name	HAL_StatusTypeDef HAL_I2S_DMAStop (I2S_HandleTypeDef * hi2s)
Function Description	Resumes the audio stream playing from the Media.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> HAL status

22.3.18 HAL_I2S_IRQHandler

Function Name	void HAL_I2S_IRQHandler (I2S_HandleTypeDef * hi2s)
Function Description	This function handles I2S interrupt request.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> None

22.3.19 HAL_I2S_TxHalfCpltCallback

Function Name	void HAL_I2S_TxHalfCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Tx Transfer Half completed callbacks.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module
Return values	<ul style="list-style-type: none"> None

22.3.20 HAL_I2S_TxCpltCallback

Function Name	void HAL_I2S_TxCpltCallback (I2S_HandleTypeDef * hi2s)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hi2s: pointer to a I2S_HandleTypeDef structure that contains

the configuration information for I2S module

Return values

- None

22.3.21 HAL_I2S_RxHalfCpltCallback

Function Name **void HAL_I2S_RxHalfCpltCallback (I2S_HandleTypeDef * hi2s)**

Function Description Rx Transfer half completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- None

22.3.22 HAL_I2S_RxCpltCallback

Function Name **void HAL_I2S_RxCpltCallback (I2S_HandleTypeDef * hi2s)**

Function Description Rx Transfer completed callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- None

22.3.23 HAL_I2S_ErrorCallback

Function Name **void HAL_I2S_ErrorCallback (I2S_HandleTypeDef * hi2s)**

Function Description I2S error callbacks.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- None

22.3.24 HAL_I2S_GetState

Function Name **HAL_I2S_StateTypeDef HAL_I2S_GetState (I2S_HandleTypeDef * hi2s)**

Function Description Return the I2S state.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- HAL state

22.3.25 HAL_I2S_GetError

Function Name **uint32_t HAL_I2S_GetError (I2S_HandleTypeDef * hi2s)**

Function Description Return the I2S error code.

Parameters

- **hi2s:** pointer to a I2S_HandleTypeDef structure that contains the configuration information for I2S module

Return values

- I2S Error Code

22.4 I2S Firmware driver defines

22.4.1 I2S

I2S Audio Frequency

I2S_AUDIOFREQ_192K

I2S_AUDIOFREQ_96K

I2S_AUDIOFREQ_48K

I2S_AUDIOFREQ_44K

I2S_AUDIOFREQ_32K

I2S_AUDIOFREQ_22K

I2S_AUDIOFREQ_16K

I2S_AUDIOFREQ_11K

I2S_AUDIOFREQ_8K

I2S_AUDIOFREQ_DEFAULT

IS_I2S_AUDIO_FREQ

I2S Clock Polarity

I2S_CPOL_LOW

I2S_CPOL_HIGH

IS_I2S_CPOL

I2S Data Format

I2S_DATAFORMAT_16B

I2S_DATAFORMAT_16B_EXTENDED

I2S_DATAFORMAT_24B

I2S_DATAFORMAT_32B

IS_I2S_DATA_FORMAT

I2S Error Codes

HAL_I2S_ERROR_NONE No error

HAL_I2S_ERROR_UDR I2S Underrun error

HAL_I2S_ERROR_OVR I2S Overrun error

HAL_I2S_ERROR_FRE I2S Frame format error

HAL_I2S_ERROR_DMA DMA transfer error

I2S Exported Macros

__HAL_I2S_RESET_HANDLE_STATE **Description:**

- Reset I2S handle state.

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

__HAL_I2S_ENABLE

- None

Description:

- Enable or disable the specified SPI peripheral (in I2S mode).

Parameters:

- __HANDLE__: specifies the I2S Handle.

Return value:

- None

__HAL_I2S_DISABLE

__HAL_I2S_ENABLE_IT

Description:

- Enable or disable the specified I2S interrupts.

Parameters:

- __HANDLE__: specifies the I2S Handle.
- __INTERRUPT__: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - I2S_IT_TXE: Tx buffer empty interrupt enable
 - I2S_IT_RXNE: RX buffer not empty interrupt enable
 - I2S_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_I2S_DISABLE_IT

__HAL_I2S_GET_IT_SOURCE

Description:

- Checks if the specified I2S interrupt source is enabled or disabled.

Parameters:

- __HANDLE__: specifies the I2S Handle. This parameter can be I2S where x: 1, 2, or 3 to select the I2S peripheral.
- __INTERRUPT__: specifies the I2S interrupt source to check. This parameter can be one of the following values:
 - I2S_IT_TXE: Tx buffer empty interrupt enable
 - I2S_IT_RXNE: RX buffer not empty interrupt enable
 - I2S_IT_ERR: Error interrupt enable

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_I2S_GET_FLAG

Description:

- Checks whether the specified I2S flag is set or not.

Parameters:

- `__HANDLE__`: specifies the I2S Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `I2S_FLAG_RXNE`: Receive buffer not empty flag
 - `I2S_FLAG_TXE`: Transmit buffer empty flag
 - `I2S_FLAG_UDR`: Underrun flag
 - `I2S_FLAG_OVR`: Overrun flag
 - `I2S_FLAG_FRE`: Frame error flag
 - `I2S_FLAG_CHSIDE`: Channel Side flag
 - `I2S_FLAG_BSY`: Busy flag

Return value:

- The: new state of `__FLAG__` (TRUE or FALSE).

Description:

- Clears the I2S OVR pending flag.

Parameters:

- `__HANDLE__`: specifies the I2S Handle.

Return value:

- None

Description:

- Clears the I2S UDR pending flag.

Parameters:

- `__HANDLE__`: specifies the I2S Handle.

Return value:

- None

`__HAL_I2S_CLEAR_OVRFLAG`

`__HAL_I2S_CLEAR_UDRFLAG`

I2S Flag definition

`I2S_FLAG_TXE`

`I2S_FLAG_RXNE`

`I2S_FLAG_UDR`

`I2S_FLAG_OVR`

`I2S_FLAG_FRE`

`I2S_FLAG_CHSIDE`

`I2S_FLAG_BSY`

I2S Interrupt configuration definition

I2S_IT_TXE

I2S_IT_RXNE

I2S_IT_ERR

I2S MCLK Output

I2S_MCLKOUTPUT_ENABLE

I2S_MCLKOUTPUT_DISABLE

IS_I2S_MCLK_OUTPUT

I2S Mode

I2S_MODE_SLAVE_TX

I2S_MODE_SLAVE_RX

I2S_MODE_MASTER_TX

I2S_MODE_MASTER_RX

IS_I2S_MODE

I2S Standard

I2S_STANDARD_PHILIPS

I2S_STANDARD_MSB

I2S_STANDARD_LSB

I2S_STANDARD_PCM_SHORT

I2S_STANDARD_PCM_LONG

IS_I2S_STANDARD

23 HAL IRDA Generic Driver

23.1 HAL IRDA Generic Driver

23.2 IRDA Firmware driver registers structures

23.2.1 IRDA_InitTypeDef

Data Fields

- *uint32_t BaudRate*
- *uint32_t WordLength*
- *uint32_t Parity*
- *uint32_t Mode*
- *uint8_t Prescaler*
- *uint32_t IrDAMode*

Field Documentation

- *uint32_t IRDA_InitTypeDef::BaudRate*
This member configures the IRDA communication baud rate. The baud rate is computed using the following formula: $\text{IntegerDivider} = ((\text{PCLKx}) / (16 * (\text{hirda->Init.BaudRate})))$ $\text{FractionalDivider} = ((\text{IntegerDivider} - ((\text{uint32_t}) \text{IntegerDivider})) * 16) + 0.5$
- *uint32_t IRDA_InitTypeDef::WordLength*
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [IRDA_Word_Length](#)
- *uint32_t IRDA_InitTypeDef::Parity*
Specifies the parity mode. This parameter can be a value of [IRDA_Parity](#)
Note:When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- *uint32_t IRDA_InitTypeDef::Mode*
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [IRDA_Transfer_Mode](#)
- *uint8_t IRDA_InitTypeDef::Prescaler*
Specifies the Prescaler value prescaler value to be programmed in the IrDA low-power Baud Register, for defining pulse width on which burst acceptance/rejection will be decided. This value is used as divisor of system clock to achieve required pulse width.
- *uint32_t IRDA_InitTypeDef::IrDAMode*
Specifies the IrDA mode This parameter can be a value of [IRDA_Low_Power](#)

23.2.2 IRDA_HandleTypeDef

Data Fields

- *USART_TypeDef * Instance*

- ***IRDA_InitTypeDef Init***
- ***uint8_t *pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***uint16_t TxXferCount***
- ***uint8_t *pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef *hdmatx***
- ***DMA_HandleTypeDef *hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_IRDA_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* IRDA_HandleTypeDef::Instance***
USART registers base address
- ***IRDA_InitTypeDef IRDA_HandleTypeDef::Init***
IRDA communication parameters
- ***uint8_t* IRDA_HandleTypeDef::pTxBuffPtr***
Pointer to IRDA Tx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::TxXferSize***
IRDA Tx Transfer size
- ***uint16_t IRDA_HandleTypeDef::TxXferCount***
IRDA Tx Transfer Counter
- ***uint8_t* IRDA_HandleTypeDef::pRxBuffPtr***
Pointer to IRDA Rx transfer Buffer
- ***uint16_t IRDA_HandleTypeDef::RxXferSize***
IRDA Rx Transfer size
- ***uint16_t IRDA_HandleTypeDef::RxXferCount***
IRDA Rx Transfer Counter
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmatx***
IRDA Tx DMA Handle parameters
- ***DMA_HandleTypeDef* IRDA_HandleTypeDef::hdmarx***
IRDA Rx DMA Handle parameters
- ***HAL_LockTypeDef IRDA_HandleTypeDef::Lock***
Locking object
- ***__IO HAL_IRDA_StateTypeDef IRDA_HandleTypeDef::State***
IRDA communication state
- ***__IO uint32_t IRDA_HandleTypeDef::ErrorCode***
IRDA Error code

23.3 IRDA Firmware driver API description

23.3.1 How to use this driver

The IRDA HAL driver can be used as follows:

1. Declare a ***IRDA_HandleTypeDef*** handle structure.
2. Initialize the IRDA low level resources by implementing the ***HAL_IRDA_MspInit()*** API:
 - a. Enable the USARTx interface clock.
 - b. IRDA pins configuration:

- Enable the clock for the IRDA GPIOs.
- Configure the IRDA pins as alternate function pull-up.
- c. NVIC configuration if you need to use interrupt process (HAL_IRDA_Transmit_IT() and HAL_IRDA_Receive_IT() APIs):
 - Configure the USARTx interrupt priority.
 - Enable the NVIC USART IRQ handle.
- d. DMA Configuration if you need to use DMA process (HAL_IRDA_Transmit_DMA() and HAL_IRDA_Receive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx channel.
 - Associate the initialized DMA handle to the IRDA DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
 - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
- 3. Program the Baud Rate, Word Length, Parity, IrDA Mode, Prescaler and Mode(Receiver/Transmitter) in the hirda Init structure.
- 4. Initialize the IRDA registers by calling the HAL_IRDA_Init() API:
 - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_IRDA_MspInit() API. The specific IRDA interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_IRDA_ENABLE_IT() and __HAL_IRDA_DISABLE_IT() inside the transmit and receive process.
- 5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_IRDA_Transmit()
- Receive an amount of data in blocking mode using HAL_IRDA_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_IRDA_Transmit_IT()
- At transmission end of transfer HAL_IRDA_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_IRDA_Receive_IT()
- At reception end of transfer HAL_IRDA_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_IRDA_Transmit_DMA()
- At transmission end of transfer HAL_IRDA_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_IRDA_TxCpltCallback

- Receive an amount of data in non blocking mode (DMA) using HAL_IRDA_Receive_DMA()
- At reception end of transfer HAL_IRDA_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_IRDA_RxCpltCallback
- In case of transfer Error, HAL_IRDA_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_IRDA_ErrorCallback

IRDA HAL driver macros list

Below the list of most used macros in IRDA HAL driver.

- __HAL_IRDA_ENABLE: Enable the IRDA peripheral
- __HAL_IRDA_DISABLE: Disable the IRDA peripheral
- __HAL_IRDA_GET_FLAG : Check whether the specified IRDA flag is set or not
- __HAL_IRDA_CLEAR_FLAG : Clear the specified IRDA pending flag
- __HAL_IRDA_ENABLE_IT: Enable the specified IRDA interrupt
- __HAL_IRDA_DISABLE_IT: Disable the specified IRDA interrupt
- __HAL_IRDA_GET_IT_SOURCE: Check whether the specified IRDA interrupt has occurred or not



You can refer to the IRDA HAL driver header file for more useful macros

23.3.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in IrDA mode.

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), the possible IRDA frame formats are as listed in [Table 18: "IRDA frame formats"](#).
 - Prescaler: A pulse of width less than two and greater than one PSC period(s) may or may not be rejected. The receiver set up time should be managed by software. The IrDA physical layer specification specifies a minimum of 10 ms delay between transmission and reception (IrDA is a half duplex protocol).
 - Mode: Receiver/transmitter modes
 - IrDAMode: the IrDA can operate in the Normal mode or in the Low power mode.

Table 18: IRDA frame formats

M bit	PCE bit	IRDA frame
0	0	SB 8 bit data STB
0	1	SB 7 bit data PB STB
1	0	SB 9 bit data STB
1	1	SB 8 bit data PB STB

The HAL_IRDA_Init() function follows IRDA configuration procedures (details for the procedures are available in reference manual (RM0038)).

This section contains the following APIs:

- [HAL_IRDA_Init\(\)](#)
- [HAL_IRDA_DeInit\(\)](#)
- [HAL_IRDA_MspInit\(\)](#)
- [HAL_IRDA_MspDeInit\(\)](#)

23.3.3 IO operation functions

This subsection provides a set of functions allowing to manage the IRDA data transfers.

IrDA is a half duplex communication protocol. If the Transmitter is busy, any data on the IrDA receive line will be ignored by the IrDA decoder and if the Receiver is busy, data on the TX from the USART to IrDA will not be encoded by IrDA. While receiving data, transmission should be avoided as the data to be transmitted could be corrupted.

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA, These API's return the HAL status. The end of the data processing will be indicated through the dedicated IRDA IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_IRDA_TxCpltCallback(), HAL_IRDA_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_IRDA_ErrorCallback() user callback will be executed when a communication error is detected
2. Blocking mode APIs are :
 - HAL_IRDA_Transmit()
 - HAL_IRDA_Receive()
3. Non Blocking mode APIs with Interrupt are :
 - HAL_IRDA_Transmit_IT()
 - HAL_IRDA_Receive_IT()
 - HAL_IRDA_IRQHandler()
4. Non Blocking mode functions with DMA are :
 - HAL_IRDA_Transmit_DMA()
 - HAL_IRDA_Receive_DMA()
 - HAL_IRDA_DMAPause()
 - HAL_IRDA_DMAResume()
 - HAL_IRDA_DMAStop()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - HAL_IRDA_TxHalfCpltCallback()
 - HAL_IRDA_TxCpltCallback()
 - HAL_IRDA_RxHalfCpltCallback()
 - HAL_IRDA_RxCpltCallback()
 - HAL_IRDA_ErrorCallback()

This section contains the following APIs:

- [HAL_IRDA_Transmit\(\)](#)
- [HAL_IRDA_Receive\(\)](#)
- [HAL_IRDA_Transmit_IT\(\)](#)
- [HAL_IRDA_Receive_IT\(\)](#)
- [HAL_IRDA_Transmit_DMA\(\)](#)
- [HAL_IRDA_Receive_DMA\(\)](#)
- [HAL_IRDA_DMAPause\(\)](#)
- [HAL_IRDA_DMAResume\(\)](#)

- [HAL_IRDA_DMAStop\(\)](#)
- [HAL_IRDA_IRQHandler\(\)](#)
- [HAL_IRDA_TxCpltCallback\(\)](#)
- [HAL_IRDA_TxHalfCpltCallback\(\)](#)
- [HAL_IRDA_RxCpltCallback\(\)](#)
- [HAL_IRDA_RxHalfCpltCallback\(\)](#)
- [HAL_IRDA_ErrorCallback\(\)](#)

23.3.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of IrDA communication process and also return Peripheral Errors occurred during communication process

- `HAL_IRDA_GetState()` API can be helpful to check in run-time the state of the IRDA peripheral.
- `HAL_IRDA_GetError()` check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- [HAL_IRDA_GetState\(\)](#)
- [HAL_IRDA_GetError\(\)](#)

23.3.5 HAL_IRDA_Init

Function Name	HAL_StatusTypeDef HAL_IRDA_Init (IRDA_HandleTypeDef * hirda)
Function Description	Initializes the IRDA mode according to the specified parameters in the <code>IRDA_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a <code>IRDA_HandleTypeDef</code> structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • HAL status

23.3.6 HAL_IRDA_DeInit

Function Name	HAL_StatusTypeDef HAL_IRDA_DeInit (IRDA_HandleTypeDef * hirda)
Function Description	DeInitializes the IRDA peripheral.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a <code>IRDA_HandleTypeDef</code> structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> • HAL status

23.3.7 HAL_IRDA_Msplnit

Function Name	void HAL_IRDA_Msplnit (IRDA_HandleTypeDef * hirda)
Function Description	IRDA MSP Init.
Parameters	<ul style="list-style-type: none"> • hirda: Pointer to a <code>IRDA_HandleTypeDef</code> structure that contains the configuration information for the specified IRDA module.

Return values • None

23.3.8 HAL_IRDA_MspDeInit

Function Name **void HAL_IRDA_MspDeInit (IRDA_HandleTypeDef * hirda)**

Function Description IRDA MSP DeInit.

Parameters • **hirda**: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values • None

23.3.9 HAL_IRDA_Transmit

Function Name **HAL_StatusTypeDef HAL_IRDA_Transmit (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function Description Sends an amount of data in blocking mode.

Parameters • **hirda**: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
• **pData**: Pointer to data buffer
• **Size**: Amount of data to be sent
• **Timeout**: Specify timeout value

Return values • HAL status

23.3.10 HAL_IRDA_Receive

Function Name **HAL_StatusTypeDef HAL_IRDA_Receive (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function Description Receive an amount of data in blocking mode.

Parameters • **hirda**: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
• **pData**: Pointer to data buffer
• **Size**: Amount of data to be received
• **Timeout**: Specify timeout value

Return values • HAL status

23.3.11 HAL_IRDA_Transmit_IT

Function Name **HAL_StatusTypeDef HAL_IRDA_Transmit_IT (IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)**

Function Description Sends an amount of data in non-blocking mode.

Parameters • **hirda**: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
• **pData**: Pointer to data buffer

- **Size:** Amount of data to be sent
- Return values
 - HAL status

23.3.12 HAL_IRDA_Receive_IT

- Function Name **HAL_StatusTypeDef HAL_IRDA_Receive_IT**
(IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
- Function Description Receives an amount of data in non-blocking mode.
- Parameters
 - **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
 - **pData:** Pointer to data buffer
 - **Size:** Amount of data to be received
- Return values
 - HAL status

23.3.13 HAL_IRDA_Transmit_DMA

- Function Name **HAL_StatusTypeDef HAL_IRDA_Transmit_DMA**
(IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
- Function Description Sends an amount of data in non-blocking mode.
- Parameters
 - **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
 - **pData:** Pointer to data buffer
 - **Size:** Amount of data to be sent
- Return values
 - HAL status

23.3.14 HAL_IRDA_Receive_DMA

- Function Name **HAL_StatusTypeDef HAL_IRDA_Receive_DMA**
(IRDA_HandleTypeDef * hirda, uint8_t * pData, uint16_t Size)
- Function Description Receive an amount of data in non-blocking mode.
- Parameters
 - **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
 - **pData:** Pointer to data buffer
 - **Size:** Amount of data to be received
- Return values
 - HAL status
- Notes
 - When the IRDA parity is enabled (PCE = 1) the data received contain the parity bit.

23.3.15 HAL_IRDA_DMAPause

- Function Name **HAL_StatusTypeDef HAL_IRDA_DMAPause**
(IRDA_HandleTypeDef * hirda)
- Function Description Pauses the DMA Transfer.
- Parameters
 - **hirda:** Pointer to a IRDA_HandleTypeDef structure that

contains the configuration information for the specified IRDA module.

Return values

- HAL status

23.3.16 HAL_IRDA_DMAResume

Function Name **HAL_StatusTypeDef HAL_IRDA_DMAResume (IRDA_HandleTypeDef * hirda)**

Function Description Resumes the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- HAL status

23.3.17 HAL_IRDA_DMAStop

Function Name **HAL_StatusTypeDef HAL_IRDA_DMAStop (IRDA_HandleTypeDef * hirda)**

Function Description Stops the DMA Transfer.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- HAL status

23.3.18 HAL_IRDA_IRQHandler

Function Name **void HAL_IRDA_IRQHandler (IRDA_HandleTypeDef * hirda)**

Function Description This function handles IRDA interrupt request.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- None

23.3.19 HAL_IRDA_TxCpltCallback

Function Name **void HAL_IRDA_TxCpltCallback (IRDA_HandleTypeDef * hirda)**

Function Description Tx Transfer completed callbacks.

Parameters

- **hirda:** Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.

Return values

- None

23.3.20 HAL_IRDA_TxHalfCpltCallback

Function Name **void HAL_IRDA_TxHalfCpltCallback (IRDA_HandleTypeDef * hirda)**

Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none"> None

23.3.21 HAL_IRDA_RxCpltCallback

Function Name	void HAL_IRDA_RxCpltCallback (IRDA_HandleTypeDef * hirda)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> None

23.3.22 HAL_IRDA_RxHalfCpltCallback

Function Name	void HAL_IRDA_RxHalfCpltCallback (IRDA_HandleTypeDef * hirda)
Function Description	Rx Half Transfer complete callbacks.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> None

23.3.23 HAL_IRDA_ErrorCallback

Function Name	void HAL_IRDA_ErrorCallback (IRDA_HandleTypeDef * hirda)
Function Description	IRDA error callbacks.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> None

23.3.24 HAL_IRDA_GetState

Function Name	HAL_IRDA_StateTypeDef HAL_IRDA_GetState (IRDA_HandleTypeDef * hirda)
Function Description	Returns the IRDA state.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> HAL state

23.3.25 HAL_IRDA_GetError

Function Name	uint32_t HAL_IRDA_GetError (IRDA_HandleTypeDef * hirda)
Function Description	Return the IRDA error code.
Parameters	<ul style="list-style-type: none"> hirda: Pointer to a IRDA_HandleTypeDef structure that contains the configuration information for the specified IRDA module.
Return values	<ul style="list-style-type: none"> IRDA Error Code

23.4 IRDA Firmware driver defines

23.4.1 IRDA

IRDA Error Codes

HAL_IRDA_ERROR_NONE	No error
HAL_IRDA_ERROR_PE	Parity error
HAL_IRDA_ERROR_NE	Noise error
HAL_IRDA_ERROR_FE	frame error
HAL_IRDA_ERROR_ORE	Overrun error
HAL_IRDA_ERROR_DMA	DMA transfer error

IRDA Exported Macros

`__HAL_IRDA_RESET_HANDLE_STATE`

Description:

- Reset IRDA handle state.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None

`__HAL_IRDA_FLUSH_DRREGISTER`

Description:

- Flush the IRDA DR register.

Parameters:

- `__HANDLE__`: specifies the USART Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

`__HAL_IRDA_GET_FLAG`

Description:

- Check whether the specified IRDA flag is set or not.

Parameters:

- `__HANDLE__`: specifies the IRDA

Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

- **__FLAG__**: specifies the flag to check. This parameter can be one of the following values:
 - IRDA_FLAG_TXE: Transmit data register empty flag
 - IRDA_FLAG_TC: Transmission Complete flag
 - IRDA_FLAG_RXNE: Receive data register not empty flag
 - IRDA_FLAG_IDLE: Idle Line detection flag
 - IRDA_FLAG_ORE: OverRun Error flag
 - IRDA_FLAG_NE: Noise Error flag
 - IRDA_FLAG_FE: Framing Error flag
 - IRDA_FLAG_PE: Parity Error flag

Return value:

- The: new state of **__FLAG__** (TRUE or FALSE).

Description:

- Clear the specified IRDA pending flag.

Parameters:

- **__HANDLE__**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- **__FLAG__**: specifies the flag to check. This parameter can be any combination of the following values:
 - IRDA_FLAG_TC: Transmission Complete flag.
 - IRDA_FLAG_RXNE: Receive data register not empty flag.

Return value:

- None

Notes:

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by software

__HAL_IRDA_CLEAR_FLAG

sequence: a read operation to USART_SR register followed by a read operation to USART_DR register. RXNE flag can be also cleared by a read to the USART_DR register. TC flag can be also cleared by software sequence: a read operation to USART_SR register followed by a write operation to USART_DR register. TXE flag is cleared only by a write to the USART_DR register.

__HAL_IRDA_CLEAR_PEFLAG

Description:

- Clear the IRDA PE pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None

__HAL_IRDA_CLEAR_FEFLAG

Description:

- Clear the IRDA FE pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None

__HAL_IRDA_CLEAR_NEFLAG

Description:

- Clear the IRDA NE pending flag.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None

__HAL_IRDA_CLEAR_OREFLAG

Description:

- Clear the IRDA ORE pending flag.

__HAL_IRDA_CLEAR_IDLEFLAG

Parameters:

- **__HANDLE__**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None

Description:

- Clear the IRDA IDLE pending flag.

Parameters:

- **__HANDLE__**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None

__HAL_IRDA_ENABLE_IT

Description:

- Enable the specified IRDA interrupt.

Parameters:

- **__HANDLE__**: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **__INTERRUPT__**: specifies the IRDA interrupt source to enable. This parameter can be one of the following values:
 - IRDA_IT_TXE: Transmit Data Register empty interrupt
 - IRDA_IT_TC: Transmission complete interrupt
 - IRDA_IT_RXNE: Receive Data register not empty interrupt
 - IRDA_IT_IDLE: Idle line detection interrupt
 - IRDA_IT_PE: Parity Error interrupt
 - IRDA_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

__HAL_IRDA_DISABLE_IT

Description:

- Disable the specified IRDA interrupt.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- `__INTERRUPT__`: specifies the IRDA interrupt source to disable. This parameter can be one of the following values:
 - `IRDA_IT_TXE`: Transmit Data Register empty interrupt
 - `IRDA_IT_TC`: Transmission complete interrupt
 - `IRDA_IT_RXNE`: Receive Data register not empty interrupt
 - `IRDA_IT_IDLE`: Idle line detection interrupt
 - `IRDA_IT_PE`: Parity Error interrupt
 - `IRDA_IT_ERR`: Error interrupt (Frame error, noise error, overrun error)

Return value:

- None

`__HAL_IRDA_GET_IT_SOURCE`**Description:**

- Check whether the specified IRDA interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- `__IT__`: specifies the IRDA interrupt source to check. This parameter can be one of the following values:
 - `IRDA_IT_TXE`: Transmit Data Register empty interrupt
 - `IRDA_IT_TC`: Transmission complete interrupt
 - `IRDA_IT_RXNE`: Receive Data register not empty interrupt
 - `IRDA_IT_IDLE`: Idle line detection interrupt
 - `IRDA_IT_ERR`: Error interrupt
 - `IRDA_IT_PE`: Parity Error interrupt

`__HAL_IRDA_ONE_BIT_SAMPLE_ENABLE`

Return value:

- The: new state of `__IT__` (TRUE or FALSE).

Description:

- Enables the IRDA one bit sample method.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

`__HAL_IRDA_ONE_BIT_SAMPLE_DISABLE`

Description:

- Disables the IRDA one bit sample method.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle.

Return value:

- None

`__HAL_IRDA_ENABLE`

Description:

- Enable UART/USART associated to IRDA Handle.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

Return value:

- None

`__HAL_IRDA_DISABLE`

Description:

- Disable UART/USART associated to IRDA Handle.

Parameters:

- `__HANDLE__`: specifies the IRDA Handle. IRDA Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

Return value:

- None

IRDA Flags

IRDA_FLAG_TXE

IRDA_FLAG_TC

IRDA_FLAG_RXNE

IRDA_FLAG_IDLE

IRDA_FLAG_ORE

IRDA_FLAG_NE

IRDA_FLAG_FE

IRDA_FLAG_PE

IRDA Interrupt Definitions

IRDA_IT_PE

IRDA_IT_TXE

IRDA_IT_TC

IRDA_IT_RXNE

IRDA_IT_IDLE

IRDA_IT_LBD

IRDA_IT_CTS

IRDA_IT_ERR

IRDA Low Power

IRDA_POWERMODE_LOWPOWER

IRDA_POWERMODE_NORMAL

IRDA One Bit Sampling

IRDA_ONE_BIT_SAMPLE_DISABLE

IRDA_ONE_BIT_SAMPLE_ENABLE

IRDA Parity

IRDA_PARITY_NONE

IRDA_PARITY_EVEN

IRDA_PARITY_ODD

IRDA Private Constants

IRDA_DR_MASK_U16_8DATABITS

IRDA_DR_MASK_U16_9DATABITS

IRDA_DR_MASK_U8_7DATABITS

IRDA_DR_MASK_U8_8DATABITS

IRDA Private Macros

IRDA_CR1_REG_INDEX

IRDA_CR2_REG_INDEX

IRDA_CR3_REG_INDEX

IRDA_DIV

IRDA_DIVMANT

IRDA_DIVFRAQ

IRDA_BRR

IS_IRDA_BAUDRATE

The maximum Baud Rate is 115200bps Returns : True or False

IS_IRDA_WORD_LENGTH

IS_IRDA_PARITY

IS_IRDA_MODE

IS_IRDA_POWERMODE

IRDA_IT_MASK

IRDA Transfer Mode

IRDA_MODE_RX

IRDA_MODE_TX

IRDA_MODE_TX_RX

IRDA Word Length

IRDA_WORDLENGTH_8B

IRDA_WORDLENGTH_9B

24 HAL IWDG Generic Driver

24.1 HAL IWDG Generic Driver

24.2 IWDG Firmware driver registers structures

24.2.1 IWDG_InitTypeDef

Data Fields

- *uint32_t Prescaler*
- *uint32_t Reload*

Field Documentation

- *uint32_t IWDG_InitTypeDef::Prescaler*
Select the prescaler of the IWDG. This parameter can be a value of [IWDG_Prescaler](#)
- *uint32_t IWDG_InitTypeDef::Reload*
Specifies the IWDG down-counter reload value. This parameter must be a number between Min_Data = 0 and Max_Data = 0xFFFF

24.2.2 IWDG_HandleTypeDef

Data Fields

- *IWDG_TypeDef * Instance*
- *IWDG_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_IWDG_StateTypeDef State*

Field Documentation

- *IWDG_TypeDef* IWDG_HandleTypeDef::Instance*
Register base address
- *IWDG_InitTypeDef IWDG_HandleTypeDef::Init*
IWDG required parameters
- *HAL_LockTypeDef IWDG_HandleTypeDef::Lock*
IWDG Locking object
- *__IO HAL_IWDG_StateTypeDef IWDG_HandleTypeDef::State*
IWDG communication state

24.3 IWDG Firmware driver API description

24.3.1 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the IWDG according to the specified parameters in the IWDG_InitTypeDef and create the associated handle
- Initialize the IWDG MSP
- DeInitialize IWDG MSP

This section contains the following APIs:

- [HAL_IWDG_Init\(\)](#)
- [HAL_IWDG_Msplnit\(\)](#)

24.3.2 IO operation functions

This section provides functions allowing to:

- Start the IWDG.
- Refresh the IWDG.

This section contains the following APIs:

- [HAL_IWDG_Start\(\)](#)
- [HAL_IWDG_Refresh\(\)](#)

24.3.3 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- [HAL_IWDG_GetState\(\)](#)

24.3.4 HAL_IWDG_Init

Function Name	HAL_StatusTypeDef HAL_IWDG_Init (IWDG_HandleTypeDef * hiwdg)
Function Description	Initializes the IWDG according to the specified parameters in the IWDG_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

24.3.5 HAL_IWDG_Msplnit

Function Name	void HAL_IWDG_Msplnit (IWDG_HandleTypeDef * hiwdg)
Function Description	Initializes the IWDG MSP.
Parameters	<ul style="list-style-type: none"> • hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> • None

24.3.6 HAL_IWDG_Start

Function Name	HAL_StatusTypeDef HAL_IWDG_Start (IWDG_HandleTypeDef * hiwdg)
Function Description	Starts the IWDG.

Parameters	<ul style="list-style-type: none"> hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> HAL status

24.3.7 HAL_IWDG_Refresh

Function Name	HAL_StatusTypeDef HAL_IWDG_Refresh (IWDG_HandleTypeDef * hiwdg)
Function Description	Refreshes the IWDG.
Parameters	<ul style="list-style-type: none"> hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> HAL status

24.3.8 HAL_IWDG_GetState

Function Name	HAL_IWDG_StateTypeDef HAL_IWDG_GetState (IWDG_HandleTypeDef * hiwdg)
Function Description	Returns the IWDG state.
Parameters	<ul style="list-style-type: none"> hiwdg: pointer to a IWDG_HandleTypeDef structure that contains the configuration information for the specified IWDG module.
Return values	<ul style="list-style-type: none"> HAL state

24.4 IWDG Firmware driver defines

24.4.1 IWDG

IWDG Exported Macros

__HAL_IWDG_RESET_HANDLE_STATE	Description: <ul style="list-style-type: none"> Reset IWDG handle state. Parameters: <ul style="list-style-type: none"> __HANDLE__: IWDG handle. Return value: <ul style="list-style-type: none"> None
__HAL_IWDG_START	Description: <ul style="list-style-type: none"> Enables the IWDG peripheral. Parameters: <ul style="list-style-type: none"> __HANDLE__: IWDG handle Return value: <ul style="list-style-type: none"> None
__HAL_IWDG_RELOAD_COUNTER	Description:

__HAL_IWDG_GET_FLAG

- Reloads IWDG counter with value defined in the reload register (write access to IWDG_PR and IWDG_RLR registers disabled).

Parameters:

- **__HANDLE__**: IWDG handle

Return value:

- None

Description:

- Gets the selected IWDG's flag status.

Parameters:

- **__HANDLE__**: IWDG handle
- **__FLAG__**: specifies the flag to check. This parameter can be one of the following values:
 - IWDG_FLAG_PVU: Watchdog counter reload value update flag
 - IWDG_FLAG_RVU: Watchdog counter prescaler value flag

Return value:

- The: new state of **__FLAG__** (TRUE or FALSE).

IWDG Prescaler

IWDG_PRESCALER_4	IWDG prescaler set to 4
IWDG_PRESCALER_8	IWDG prescaler set to 8
IWDG_PRESCALER_16	IWDG prescaler set to 16
IWDG_PRESCALER_32	IWDG prescaler set to 32
IWDG_PRESCALER_64	IWDG prescaler set to 64
IWDG_PRESCALER_128	IWDG prescaler set to 128
IWDG_PRESCALER_256	IWDG prescaler set to 256

IWDG Private Defines

HAL_IWDG_DEFAULT_TIMEOUT

IWDG_KEY_RELOAD IWDG Reload Counter Enable

IWDG_KEY_ENABLE IWDG Peripheral Enable

IWDG_KEY_WRITE_ACCESS_ENABLE IWDG KR Write Access Enable

IWDG_KEY_WRITE_ACCESS_DISABLE IWDG KR Write Access Disable

IWDG_FLAG_PVU Watchdog counter prescaler value update flag

IWDG_FLAG_RVU Watchdog counter reload value update flag

IWDG Private MacrosIWDG_ENABLE_WRITE_ACCESS **Description:**

- Enables write access to IWDG_PR and IWDG_RLR registers.

Parameters:

- `__HANDLE__`: IWDG handle

Return value:

- None

IWDG_DISABLE_WRITE_ACCESS**Description:**

- Disables write access to IWDG_PR and IWDG_RLR registers.

Parameters:

- `__HANDLE__`: IWDG handle

Return value:

- None

IS_IWDG_PRESCALER**Description:**

- Check IWDG prescaler value.

Parameters:

- `__PRESCALER__`: IWDG prescaler value

Return value:

- None

IS_IWDG_RELOAD**Description:**

- Check IWDG reload value.

Parameters:

- `__RELOAD__`: IWDG reload value

Return value:

- None

25 HAL LCD Generic Driver

25.1 HAL LCD Generic Driver

25.2 LCD Firmware driver registers structures

25.2.1 LCD_InitTypeDef

Data Fields

- *uint32_t Prescaler*
- *uint32_t Divider*
- *uint32_t Duty*
- *uint32_t Bias*
- *uint32_t VoltageSource*
- *uint32_t Contrast*
- *uint32_t DeadTime*
- *uint32_t PulseOnDuration*
- *uint32_t HighDrive*
- *uint32_t BlinkMode*
- *uint32_t BlinkFrequency*
- *uint32_t MuxSegment*

Field Documentation

- *uint32_t LCD_InitTypeDef::Prescaler*
Configures the LCD Prescaler. This parameter can be one value of [LCD_Prescaler](#)
- *uint32_t LCD_InitTypeDef::Divider*
Configures the LCD Divider. This parameter can be one value of [LCD_Divider](#)
- *uint32_t LCD_InitTypeDef::Duty*
Configures the LCD Duty. This parameter can be one value of [LCD_Duty](#)
- *uint32_t LCD_InitTypeDef::Bias*
Configures the LCD Bias. This parameter can be one value of [LCD_Bias](#)
- *uint32_t LCD_InitTypeDef::VoltageSource*
Selects the LCD Voltage source. This parameter can be one value of [LCD_Voltage_Source](#)
- *uint32_t LCD_InitTypeDef::Contrast*
Configures the LCD Contrast. This parameter can be one value of [LCD_Contrast](#)
- *uint32_t LCD_InitTypeDef::DeadTime*
Configures the LCD Dead Time. This parameter can be one value of [LCD_DeadTime](#)
- *uint32_t LCD_InitTypeDef::PulseOnDuration*
Configures the LCD Pulse On Duration. This parameter can be one value of [LCD_PulseOnDuration](#)
- *uint32_t LCD_InitTypeDef::HighDrive*
Configures the LCD High Drive. This parameter can be one value of [LCD_HighDrive](#)
- *uint32_t LCD_InitTypeDef::BlinkMode*
Configures the LCD Blink Mode. This parameter can be one value of [LCD_BlinkMode](#)

- ***uint32_t LCD_InitTypeDef::BlinkFrequency***
Configures the LCD Blink frequency. This parameter can be one value of [LCD_BlinkFrequency](#)
- ***uint32_t LCD_InitTypeDef::MuxSegment***
Enable or disable mux segment. This parameter can be set to ENABLE or DISABLE.

25.2.2 LCD_HandleTypeDef

Data Fields

- ***LCD_TypeDef * Instance***
- ***LCD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_LCD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***LCD_TypeDef* LCD_HandleTypeDef::Instance***
- ***LCD_InitTypeDef LCD_HandleTypeDef::Init***
- ***HAL_LockTypeDef LCD_HandleTypeDef::Lock***
- ***__IO HAL_LCD_StateTypeDef LCD_HandleTypeDef::State***
- ***__IO uint32_t LCD_HandleTypeDef::ErrorCode***

25.3 LCD Firmware driver API description

25.3.1 How to use this driver

The LCD HAL driver can be used as follows:

1. Declare a LCD_HandleTypeDef handle structure.
2. Initialize the LCD low level resources by implement the HAL_LCD_MspInit() API:
 - a. Enable the LCDCLK (same as RTCCLK): to configure the RTCCLK/LCDCLK, proceed as follows:
 - Use RCC function HAL_RCCEx_PeriphCLKConfig in indicating RCC_PERIPHCLK_LCD and selected clock source (HSE, LSI or LSE)
 - The frequency generator allows you to achieve various LCD frame rates starting from an LCD input clock frequency (LCDCLK) which can vary from 32 kHz up to 1 MHz.
 - b. LCD pins configuration:
 - Enable the clock for the LCD GPIOs.
 - Configure these LCD pins as alternate function no-pull.
 - c. Enable the LCD interface clock.
3. Program the Prescaler, Divider, Blink mode, Blink Frequency Duty, Bias, Voltage Source, Dead Time, Pulse On Duration and Contrast in the hlcd Init structure.
4. Initialize the LCD registers by calling the HAL_LCD_Init() API. The HAL_LCD_Init() API configures also the low level Hardware GPIO, CLOCK, ...etc) by calling the customized HAL_LCD_MspInit() API. After calling the HAL_LCD_Init() the LCD RAM memory is cleared
5. Optionally you can update the LCD configuration using these macros:

- LCD High Drive using the `__HAL_LCD_HIGHDRIVER_ENABLE()` and `__HAL_LCD_HIGHDRIVER_DISABLE()` macros
 - LCD Pulse ON Duration using the `__HAL_LCD_PULSEONDURATION_CONFIG()` macro
 - LCD Dead Time using the `__HAL_LCD_DEADTIME_CONFIG()` macro
 - The LCD Blink mode and frequency using the `__HAL_LCD_BLINK_CONFIG()` macro
 - The LCD Contrast using the `__HAL_LCD_CONTRAST_CONFIG()` macro
6. Write to the LCD RAM memory using the `HAL_LCD_Write()` API, this API can be called more time to update the different LCD RAM registers before calling `HAL_LCD_UpdateDisplayRequest()` API.
 7. The `HAL_LCD_Clear()` API can be used to clear the LCD RAM memory.
 8. When LCD RAM memory is updated enable the update display request using the `HAL_LCD_UpdateDisplayRequest()` API.

LCD and low power modes:

1. The LCD remain active during STOP mode.

25.3.2 Initialization and Configuration functions

This section contains the following APIs:

- [`HAL_LCD_DeInit\(\)`](#)
- [`HAL_LCD_Init\(\)`](#)
- [`HAL_LCD_MspDeInit\(\)`](#)
- [`HAL_LCD_MspInit\(\)`](#)

25.3.3 IO operation functions

Using its double buffer memory the LCD controller ensures the coherency of the displayed information without having to use interrupts to control LCD_RAM modification.

- The application software can access the first buffer level (LCD_RAM) through the APB interface. Once it has modified the LCD_RAM using the `HAL_LCD_Write()` API, it sets the UDR flag in the LCD_SR register using the `HAL_LCD_UpdateDisplayRequest()` API. This UDR flag (update display request) requests the updated information to be moved into the second buffer level (LCD_DISPLAY).
- This operation is done synchronously with the frame (at the beginning of the next frame), until the update is completed, the LCD_RAM is write protected and the UDR flag stays high.
- Once the update is completed another flag (UDD - Update Display Done) is set and generates an interrupt if the UDDIE bit in the LCD_FCR register is set. The time it takes to update LCD_DISPLAY is, in the worst case, one odd and one even frame.
- The update will not occur (UDR = 1 and UDD = 0) until the display is enabled (LCDEN = 1).

This section contains the following APIs:

- [`HAL_LCD_Write\(\)`](#)
- [`HAL_LCD_Clear\(\)`](#)
- [`HAL_LCD_UpdateDisplayRequest\(\)`](#)

25.3.4 Peripheral State functions

This subsection provides a set of functions allowing to control the LCD:

- HAL_LCD_GetState() API can be helpful to check in run-time the state of the LCD peripheral State.
- HAL_LCD_GetError() API to return the LCD error code.

This section contains the following APIs:

- [HAL_LCD_GetState\(\)](#)
- [HAL_LCD_GetError\(\)](#)

25.3.5 HAL_LCD_DeInit

Function Name	HAL_StatusTypeDef HAL_LCD_DeInit (LCD_HandleTypeDef * hlcd)
Function Description	DeInitializes the LCD peripheral.
Parameters	<ul style="list-style-type: none"> • hlcd: LCD handle
Return values	<ul style="list-style-type: none"> • HAL status

25.3.6 HAL_LCD_Init

Function Name	HAL_StatusTypeDef HAL_LCD_Init (LCD_HandleTypeDef * hlcd)
Function Description	Initializes the LCD peripheral according to the specified parameters in the LCD_InitStruct.
Parameters	<ul style="list-style-type: none"> • hlcd: LCD handle
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This function can be used only when the LCD is disabled. The LCD HighDrive can be enabled/disabled using related macros up to user.

25.3.7 HAL_LCD_MspDeInit

Function Name	void HAL_LCD_MspDeInit (LCD_HandleTypeDef * hlcd)
Function Description	LCD MSP DeInit.
Parameters	<ul style="list-style-type: none"> • hlcd: LCD handle
Return values	<ul style="list-style-type: none"> • None

25.3.8 HAL_LCD_MspInit

Function Name	void HAL_LCD_MspInit (LCD_HandleTypeDef * hlcd)
Function Description	LCD MSP Init.
Parameters	<ul style="list-style-type: none"> • hlcd: LCD handle
Return values	<ul style="list-style-type: none"> • None

25.3.9 HAL_LCD_Write

Function Name	HAL_StatusTypeDef HAL_LCD_Write (LCD_HandleTypeDef * hlcd, uint32_t RAMRegisterIndex, uint32_t RAMRegisterMask, uint32_t Data)
---------------	---

Function Description	Writes a word in the specific LCD RAM.
Parameters	<ul style="list-style-type: none"> • hlcd: LCD handle • RAMRegisterIndex: specifies the LCD RAM Register. This parameter can be one of the following values: LCD_RAM_REGISTER0: LCD RAM Register 0LCD_RAM_REGISTER1: LCD RAM Register 1LCD_RAM_REGISTER2: LCD RAM Register 2LCD_RAM_REGISTER3: LCD RAM Register 3LCD_RAM_REGISTER4: LCD RAM Register 4LCD_RAM_REGISTER5: LCD RAM Register 5LCD_RAM_REGISTER6: LCD RAM Register 6LCD_RAM_REGISTER7: LCD RAM Register 7LCD_RAM_REGISTER8: LCD RAM Register 8LCD_RAM_REGISTER9: LCD RAM Register 9LCD_RAM_REGISTER10: LCD RAM Register 10LCD_RAM_REGISTER11: LCD RAM Register 11LCD_RAM_REGISTER12: LCD RAM Register 12LCD_RAM_REGISTER13: LCD RAM Register 13LCD_RAM_REGISTER14: LCD RAM Register 14LCD_RAM_REGISTER15: LCD RAM Register 15 • RAMRegisterMask: specifies the LCD RAM Register Data Mask. • Data: specifies LCD Data Value to be written.
Return values	<ul style="list-style-type: none"> • None

25.3.10 HAL_LCD_Clear

Function Name	HAL_StatusTypeDef HAL_LCD_Clear (LCD_HandleTypeDef * hlcd)
Function Description	Clears the LCD RAM registers.
Parameters	<ul style="list-style-type: none"> • hlcd: LCD handle
Return values	<ul style="list-style-type: none"> • None

25.3.11 HAL_LCD_UpdateDisplayRequest

Function Name	HAL_StatusTypeDef HAL_LCD_UpdateDisplayRequest (LCD_HandleTypeDef * hlcd)
Function Description	Enables the Update Display Request.
Parameters	<ul style="list-style-type: none"> • hlcd: LCD handle
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • Each time software modifies the LCD_RAM it must set the UDR bit to transfer the updated data to the second level buffer. The UDR bit stays set until the end of the update and during this time the LCD_RAM is write protected. • When the display is disabled, the update is performed for all LCD_DISPLAY locations. When the display is enabled, the update is performed only for locations for which commons are active (depending on DUTY). For example if DUTY = 1/2, only the LCD_DISPLAY of COM0 and COM1 will be updated.

25.3.12 HAL_LCD_GetState

Function Name	HAL_LCD_StateTypeDef HAL_LCD_GetState (LCD_HandleTypeDef * hlcd)
Function Description	Returns the LCD state.
Parameters	<ul style="list-style-type: none"> hlcd: LCD handle
Return values	<ul style="list-style-type: none"> HAL state

25.3.13 HAL_LCD_GetError

Function Name	uint32_t HAL_LCD_GetError (LCD_HandleTypeDef * hlcd)
Function Description	Return the LCD error code.
Parameters	<ul style="list-style-type: none"> hlcd: LCD handle
Return values	<ul style="list-style-type: none"> LCD Error Code

25.4 LCD Firmware driver defines**25.4.1 LCD*****LCD Bias***

LCD_BIAS_1_4	1/4 Bias
LCD_BIAS_1_2	1/2 Bias
LCD_BIAS_1_3	1/3 Bias
IS_LCD_BIAS	

LCD Blink Frequency

LCD_BLINKFREQUENCY_DIV8	The Blink frequency = fLCD/8
LCD_BLINKFREQUENCY_DIV16	The Blink frequency = fLCD/16
LCD_BLINKFREQUENCY_DIV32	The Blink frequency = fLCD/32
LCD_BLINKFREQUENCY_DIV64	The Blink frequency = fLCD/64
LCD_BLINKFREQUENCY_DIV128	The Blink frequency = fLCD/128
LCD_BLINKFREQUENCY_DIV256	The Blink frequency = fLCD/256
LCD_BLINKFREQUENCY_DIV512	The Blink frequency = fLCD/512
LCD_BLINKFREQUENCY_DIV1024	The Blink frequency = fLCD/1024
IS_LCD_BLINK_FREQUENCY	

LCD Blink Mode

LCD_BLINKMODE_OFF	Blink disabled
LCD_BLINKMODE_SEG0_COM0	Blink enabled on SEG[0], COM[0] (1 pixel)
LCD_BLINKMODE_SEG0_ALLCOM	Blink enabled on SEG[0], all COM (up to 8 pixels according to the programmed duty)
LCD_BLINKMODE_ALLSEG_ALLCOM	Blink enabled on all SEG and all COM (all pixels)
IS_LCD_BLINK_MODE	

LCD Contrast

LCD_CONTRASTLEVEL_0	Maximum Voltage = 2.60V
LCD_CONTRASTLEVEL_1	Maximum Voltage = 2.73V
LCD_CONTRASTLEVEL_2	Maximum Voltage = 2.86V
LCD_CONTRASTLEVEL_3	Maximum Voltage = 2.99V
LCD_CONTRASTLEVEL_4	Maximum Voltage = 3.12V
LCD_CONTRASTLEVEL_5	Maximum Voltage = 3.25V
LCD_CONTRASTLEVEL_6	Maximum Voltage = 3.38V
LCD_CONTRASTLEVEL_7	Maximum Voltage = 3.51V

IS_LCD_CONTRAST

LCD Dead Time

LCD_DEADTIME_0	No dead Time
LCD_DEADTIME_1	One Phase between different couple of Frame
LCD_DEADTIME_2	Two Phase between different couple of Frame
LCD_DEADTIME_3	Three Phase between different couple of Frame
LCD_DEADTIME_4	Four Phase between different couple of Frame
LCD_DEADTIME_5	Five Phase between different couple of Frame
LCD_DEADTIME_6	Six Phase between different couple of Frame
LCD_DEADTIME_7	Seven Phase between different couple of Frame

IS_LCD_DEAD_TIME

LCD Divider

LCD_DIVIDER_16	LCD frequency = CLKPS/16
LCD_DIVIDER_17	LCD frequency = CLKPS/17
LCD_DIVIDER_18	LCD frequency = CLKPS/18
LCD_DIVIDER_19	LCD frequency = CLKPS/19
LCD_DIVIDER_20	LCD frequency = CLKPS/20
LCD_DIVIDER_21	LCD frequency = CLKPS/21
LCD_DIVIDER_22	LCD frequency = CLKPS/22
LCD_DIVIDER_23	LCD frequency = CLKPS/23
LCD_DIVIDER_24	LCD frequency = CLKPS/24
LCD_DIVIDER_25	LCD frequency = CLKPS/25
LCD_DIVIDER_26	LCD frequency = CLKPS/26
LCD_DIVIDER_27	LCD frequency = CLKPS/27
LCD_DIVIDER_28	LCD frequency = CLKPS/28
LCD_DIVIDER_29	LCD frequency = CLKPS/29
LCD_DIVIDER_30	LCD frequency = CLKPS/30

LCD_DIVIDER_31 LCD frequency = CLKPS/31

IS_LCD_DIVIDER

LCD Duty

LCD_DUTY_STATIC Static duty

LCD_DUTY_1_2 1/2 duty

LCD_DUTY_1_3 1/3 duty

LCD_DUTY_1_4 1/4 duty

LCD_DUTY_1_8 1/8 duty

IS_LCD_DUTY

LCD Error Codes

HAL_LCD_ERROR_NONE No error

HAL_LCD_ERROR_FCRSF Synchro flag timeout error

HAL_LCD_ERROR_UDR Update display request flag timeout error

HAL_LCD_ERROR_UDD Update display done flag timeout error

HAL_LCD_ERROR_ENS LCD enabled status flag timeout error

HAL_LCD_ERROR_RDY LCD Booster ready timeout error

LCD Exported Macros

__HAL_LCD_RESET_HANDLE_STATE **Description:**

- Reset LCD handle state.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_ENABLE

Description:

- macros to enables or disables the LCD

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

__HAL_LCD_DISABLE

__HAL_LCD_HIGHDRIVER_ENABLE

Description:

- Macros to enable or disable the low resistance divider.

Parameters:

- __HANDLE__: specifies the LCD Handle.

Return value:

- None

Notes:

- When this mode is enabled, the PulseOn Duration (PON) have to be programmed to $1/CK_PS$ (LCD_PULSEONDURATION_1).

__HAL_LCD_HIGHDRIVER_DISABLE
__HAL_LCD_PULSEONDURATION_CONFIG

Description:

- Macro to configure the LCD pulses on duration.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __DURATION__: specifies the LCD pulse on duration in terms of CK_PS (prescaled LCD clock period) pulses. This parameter can be one of the following values:
 - LCD_PULSEONDURATION_0: 0 pulse
 - LCD_PULSEONDURATION_1: Pulse ON duration = $1/CK_PS$
 - LCD_PULSEONDURATION_2: Pulse ON duration = $2/CK_PS$
 - LCD_PULSEONDURATION_3: Pulse ON duration = $3/CK_PS$
 - LCD_PULSEONDURATION_4: Pulse ON duration = $4/CK_PS$
 - LCD_PULSEONDURATION_5: Pulse ON duration = $5/CK_PS$
 - LCD_PULSEONDURATION_6: Pulse ON duration = $6/CK_PS$
 - LCD_PULSEONDURATION_7: Pulse ON duration = $7/CK_PS$

Return value:

- None

__HAL_LCD_DEADTIME_CONFIG

Description:

- Macro to configure the LCD dead time.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __DEADTIME__: specifies the LCD dead time. This parameter can be one of the following values:
 - LCD_DEADTIME_0: No dead Time
 - LCD_DEADTIME_1: One Phase between different couple of Frame
 - LCD_DEADTIME_2: Two Phase between different couple of Frame
 - LCD_DEADTIME_3: Three Phase between different couple of Frame

- LCD_DEADTIME_4: Four Phase between different couple of Frame
- LCD_DEADTIME_5: Five Phase between different couple of Frame
- LCD_DEADTIME_6: Six Phase between different couple of Frame
- LCD_DEADTIME_7: Seven Phase between different couple of Frame

Return value:

- None

__HAL_LCD_CONTRAST_CONFIG**Description:**

- Macro to configure the LCD Contrast.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __CONTRAST__: specifies the LCD Contrast. This parameter can be one of the following values:
 - LCD_CONTRASTLEVEL_0: Maximum Voltage = 2.60V
 - LCD_CONTRASTLEVEL_1: Maximum Voltage = 2.73V
 - LCD_CONTRASTLEVEL_2: Maximum Voltage = 2.86V
 - LCD_CONTRASTLEVEL_3: Maximum Voltage = 2.99V
 - LCD_CONTRASTLEVEL_4: Maximum Voltage = 3.12V
 - LCD_CONTRASTLEVEL_5: Maximum Voltage = 3.25V
 - LCD_CONTRASTLEVEL_6: Maximum Voltage = 3.38V
 - LCD_CONTRASTLEVEL_7: Maximum Voltage = 3.51V

Return value:

- None

__HAL_LCD_BLINK_CONFIG**Description:**

- Macro to configure the LCD Blink mode and Blink frequency.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __BLINKMODE__: specifies the LCD blink mode. This parameter can be one of the following values:
 - LCD_BLINKMODE_OFF: Blink disabled
 - LCD_BLINKMODE_SEG0_COM0: Blink enabled on SEG[0], COM[0] (1 pixel)

- LCD_BLINKMODE_SEG0_ALLCOM: Blink enabled on SEG[0], all COM (up to 8 pixels according to the programmed duty)
- LCD_BLINKMODE_ALLSEG_ALLCOM: Blink enabled on all SEG and all COM (all pixels)
- __BLINKFREQUENCY__: specifies the LCD blink frequency.
 - LCD_BLINKFREQUENCY_DIV8: The Blink frequency = $f_{Lcd}/8$
 - LCD_BLINKFREQUENCY_DIV16: The Blink frequency = $f_{Lcd}/16$
 - LCD_BLINKFREQUENCY_DIV32: The Blink frequency = $f_{Lcd}/32$
 - LCD_BLINKFREQUENCY_DIV64: The Blink frequency = $f_{Lcd}/64$
 - LCD_BLINKFREQUENCY_DIV128: The Blink frequency = $f_{Lcd}/128$
 - LCD_BLINKFREQUENCY_DIV256: The Blink frequency = $f_{Lcd}/256$
 - LCD_BLINKFREQUENCY_DIV512: The Blink frequency = $f_{Lcd}/512$
 - LCD_BLINKFREQUENCY_DIV1024: The Blink frequency = $f_{Lcd}/1024$

Return value:

- None

Description:

- Enables or disables the specified LCD interrupt.

Parameters:

- __HANDLE__: specifies the LCD Handle.
- __INTERRUPT__: specifies the LCD interrupt source to be enabled or disabled. This parameter can be one of the following values:
 - LCD_IT_SOF: Start of Frame Interrupt
 - LCD_IT_UDD: Update Display Done Interrupt

Return value:

- None

Description:

- Checks whether the specified LCD interrupt is enabled or not.

Parameters:

- __HANDLE__: specifies the LCD Handle.

__HAL_LCD_ENABLE_IT

__HAL_LCD_DISABLE_IT

__HAL_LCD_GET_IT_SOURCE

- `__IT__`: specifies the LCD interrupt source to check. This parameter can be one of the following values:
 - `LCD_IT_SOF`: Start of Frame Interrupt
 - `LCD_IT_UDD`: Update Display Done Interrupt.

Return value:

- The: state of `__IT__` (TRUE or FALSE).

Notes:

- If the device is in STOP mode (PCLK not provided) UDD will not generate an interrupt even if `UDDIE` = 1. If the display is not enabled the UDD interrupt will never occur.

`__HAL_LCD_GET_FLAG`**Description:**

- Checks whether the specified LCD flag is set or not.

Parameters:

- `__HANDLE__`: specifies the LCD Handle.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `LCD_FLAG_ENS`: LCD Enabled flag. It indicates the LCD controller status.

Return value:

- The: new state of `__FLAG__` (TRUE or FALSE).

Notes:

- The ENS bit is set immediately when the LCDEN bit in the LCD_CR goes from 0 to 1. On deactivation it reflects the real status of LCD so it becomes 0 at the end of the last displayed frame. `LCD_FLAG_SOF`: Start of Frame flag. This flag is set by hardware at the beginning of a new frame, at the same time as the display data is updated. `LCD_FLAG_UDR`: Update Display Request flag. `LCD_FLAG_UDD`: Update Display Done flag. `LCD_FLAG_RDY`: Step-up converter Ready flag. It indicates the status of the step-up converter. `LCD_FLAG_FCRSF`: LCD Frame Control Register Synchronization Flag. This flag is set by hardware each time the LCD_FCR register is updated in the LCDCLK domain.

`__HAL_LCD_CLEAR_FLAG`**Description:**

- Clears the specified LCD pending flag.

Parameters:

- `__HANDLE__`: specifies the LCD Handle.
- `__FLAG__`: specifies the flag to clear. This parameter can be any combination of the following values:
 - `LCD_FLAG_SOF`: Start of Frame Interrupt
 - `LCD_FLAG_UDD`: Update Display Done Interrupt

Return value:

- None

LCD Flag

`LCD_FLAG_ENS`
`LCD_FLAG_SOF`
`LCD_FLAG_UDR`
`LCD_FLAG_UDD`
`LCD_FLAG_RDY`
`LCD_FLAG_FCRSF`

LCD HighDrive

`LCD_HIGHDRIVE_0` Low resistance Drive
`LCD_HIGHDRIVE_1` High resistance Drive
`IS_LCD_HIGHDRIVE`

LCD Interrupts

`LCD_IT_SOF`
`LCD_IT_UDD`

LCD Mux Segment

`LCD_MUXSEGMENT_DISABLE` SEG pin multiplexing disabled
`LCD_MUXSEGMENT_ENABLE` SEG[31:28] are multiplexed with SEG[43:40]
`IS_LCD_MUXSEGMENT`

LCD Prescaler

`LCD_PRESCALER_1` CLKPS = LCDCLK
`LCD_PRESCALER_2` CLKPS = LCDCLK/2
`LCD_PRESCALER_4` CLKPS = LCDCLK/4
`LCD_PRESCALER_8` CLKPS = LCDCLK/8
`LCD_PRESCALER_16` CLKPS = LCDCLK/16
`LCD_PRESCALER_32` CLKPS = LCDCLK/32
`LCD_PRESCALER_64` CLKPS = LCDCLK/64
`LCD_PRESCALER_128` CLKPS = LCDCLK/128

LCD_PRESCALER_256	CLKPS = LCDCLK/256
LCD_PRESCALER_512	CLKPS = LCDCLK/512
LCD_PRESCALER_1024	CLKPS = LCDCLK/1024
LCD_PRESCALER_2048	CLKPS = LCDCLK/2048
LCD_PRESCALER_4096	CLKPS = LCDCLK/4096
LCD_PRESCALER_8192	CLKPS = LCDCLK/8192
LCD_PRESCALER_16384	CLKPS = LCDCLK/16384
LCD_PRESCALER_32768	CLKPS = LCDCLK/32768

IS_LCD_PRESCALER

LCD Private Defines

LCD_TIMEOUT_VALUE

LCD Pulse On Duration

LCD_PULSEONDURATION_0	Pulse ON duration = 0 pulse
LCD_PULSEONDURATION_1	Pulse ON duration = 1/CK_PS
LCD_PULSEONDURATION_2	Pulse ON duration = 2/CK_PS
LCD_PULSEONDURATION_3	Pulse ON duration = 3/CK_PS
LCD_PULSEONDURATION_4	Pulse ON duration = 4/CK_PS
LCD_PULSEONDURATION_5	Pulse ON duration = 5/CK_PS
LCD_PULSEONDURATION_6	Pulse ON duration = 6/CK_PS
LCD_PULSEONDURATION_7	Pulse ON duration = 7/CK_PS

IS_LCD_PULSE_ON_DURATION

LCD RAMRegister

LCD_RAM_REGISTER0	LCD RAM Register 0
LCD_RAM_REGISTER1	LCD RAM Register 1
LCD_RAM_REGISTER2	LCD RAM Register 2
LCD_RAM_REGISTER3	LCD RAM Register 3
LCD_RAM_REGISTER4	LCD RAM Register 4
LCD_RAM_REGISTER5	LCD RAM Register 5
LCD_RAM_REGISTER6	LCD RAM Register 6
LCD_RAM_REGISTER7	LCD RAM Register 7
LCD_RAM_REGISTER8	LCD RAM Register 8
LCD_RAM_REGISTER9	LCD RAM Register 9
LCD_RAM_REGISTER10	LCD RAM Register 10
LCD_RAM_REGISTER11	LCD RAM Register 11
LCD_RAM_REGISTER12	LCD RAM Register 12
LCD_RAM_REGISTER13	LCD RAM Register 13

LCD_RAM_REGISTER14 LCD RAM Register 14

LCD_RAM_REGISTER15 LCD RAM Register 15

IS_LCD_RAM_REGISTER

LCD Voltage Source

LCD_VOLTAGESOURCE_INTERNAL Internal voltage source for the LCD

LCD_VOLTAGESOURCE_EXTERNAL External voltage source for the LCD

IS_LCD_VOLTAGE_SOURCE

26 HAL NOR Generic Driver

26.1 HAL NOR Generic Driver

26.2 NOR Firmware driver registers structures

26.2.1 NOR_IDTypeDef

Data Fields

- *uint16_t Manufacturer_Code*
- *uint16_t Device_Code1*
- *uint16_t Device_Code2*
- *uint16_t Device_Code3*

Field Documentation

- *uint16_t NOR_IDTypeDef::Manufacturer_Code*
Defines the device's manufacturer code used to identify the memory
- *uint16_t NOR_IDTypeDef::Device_Code1*
- *uint16_t NOR_IDTypeDef::Device_Code2*
- *uint16_t NOR_IDTypeDef::Device_Code3*
Defines the device's codes used to identify the memory. These codes can be accessed by performing read operations with specific control signals and addresses set. They can also be accessed by issuing an Auto Select command

26.2.2 NOR_CFIDTypeDef

Data Fields

- *uint16_t CFI_1*
- *uint16_t CFI_2*
- *uint16_t CFI_3*
- *uint16_t CFI_4*

Field Documentation

- *uint16_t NOR_CFIDTypeDef::CFI_1*
< Defines the information stored in the memory's Common flash interface which contains a description of various electrical and timing parameters, density information and functions supported by the memory
- *uint16_t NOR_CFIDTypeDef::CFI_2*
- *uint16_t NOR_CFIDTypeDef::CFI_3*
- *uint16_t NOR_CFIDTypeDef::CFI_4*

26.2.3 NOR_HandleTypeDef

Data Fields

- ***FSMC_NORSRAM_TypeDef * Instance***
- ***FSMC_NORSRAM_EXTENDED_TypeDef * Extended***
- ***FSMC_NORSRAM_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_NOR_StateTypeDef State***

Field Documentation

- ***FSMC_NORSRAM_TypeDef* NOR_HandleTypeDef::Instance***
Register base address
- ***FSMC_NORSRAM_EXTENDED_TypeDef* NOR_HandleTypeDef::Extended***
Extended mode register base address
- ***FSMC_NORSRAM_InitTypeDef NOR_HandleTypeDef::Init***
NOR device control configuration parameters
- ***HAL_LockTypeDef NOR_HandleTypeDef::Lock***
NOR locking object
- ***__IO HAL_NOR_StateTypeDef NOR_HandleTypeDef::State***
NOR device access state

26.3 NOR Firmware driver API description

26.3.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control NOR flash memories. It uses the FSMC layer functions to interface with NOR devices. This driver is used as follows:

- NOR flash memory configuration sequence using the function `HAL_NOR_Init()` with control and timing parameters for both normal and extended mode.
- Read NOR flash memory manufacturer code and device IDs using the function `HAL_NOR_Read_ID()`. The read information is stored in the `NOR_ID_TypeDef` structure declared by the function caller.
- Access NOR flash memory by read/write data unit operations using the functions `HAL_NOR_Read()`, `HAL_NOR_Program()`.
- Perform NOR flash erase block/chip operations using the functions `HAL_NOR_Erase_Block()` and `HAL_NOR_Erase_Chip()`.
- Read the NOR flash CFI (common flash interface) IDs using the function `HAL_NOR_Read_CFI()`. The read information is stored in the `NOR_CFI_TypeDef` structure declared by the function caller.
- You can also control the NOR device by calling the control APIs `HAL_NOR_WriteOperation_Enable()/ HAL_NOR_WriteOperation_Disable()` to respectively enable/disable the NOR write operation
- You can monitor the NOR device HAL state by calling the function `HAL_NOR_GetState()`



This driver is a set of generic APIs which handle standard NOR flash operations. If a NOR flash device contains different operations and/or implementations, it should be implemented separately.

NOR HAL driver macros list

Below the list of most used macros in NOR HAL driver.

- `NOR_WRITE` : NOR memory write data to specified address

26.3.2 NOR Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the NOR memory

This section contains the following APIs:

- [`HAL_NOR_Init\(\)`](#)
- [`HAL_NOR_DeInit\(\)`](#)
- [`HAL_NOR_MspInit\(\)`](#)
- [`HAL_NOR_MspDeInit\(\)`](#)
- [`HAL_NOR_MspWait\(\)`](#)

26.3.3 NOR Input and Output functions

This section provides functions allowing to use and control the NOR memory

This section contains the following APIs:

- [`HAL_NOR_Read_ID\(\)`](#)
- [`HAL_NOR_ReturnToReadMode\(\)`](#)
- [`HAL_NOR_Read\(\)`](#)
- [`HAL_NOR_Program\(\)`](#)
- [`HAL_NOR_ReadBuffer\(\)`](#)
- [`HAL_NOR_ProgramBuffer\(\)`](#)
- [`HAL_NOR_Erase_Block\(\)`](#)
- [`HAL_NOR_Erase_Chip\(\)`](#)
- [`HAL_NOR_Read_CFI\(\)`](#)

26.3.4 NOR Control functions

This subsection provides a set of functions allowing to control dynamically the NOR interface.

This section contains the following APIs:

- [`HAL_NOR_WriteOperation_Enable\(\)`](#)
- [`HAL_NOR_WriteOperation_Disable\(\)`](#)

26.3.5 NOR State functions

This subsection permits to get in run-time the status of the NOR controller and the data flow.

This section contains the following APIs:

- [`HAL_NOR_GetState\(\)`](#)
- [`HAL_NOR_GetStatus\(\)`](#)

26.3.6 HAL_NOR_Init

	Function Name	HAL_StatusTypeDef HAL_NOR_Init (NOR_HandleTypeDef * hnor, FSMC_NORSRAM_TimingTypeDef * Timing, FSMC_NORSRAM_TimingTypeDef * ExtTiming)
	Function Description	Perform the NOR memory Initialization sequence.
	Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • Timing: pointer to NOR control timing structure • ExtTiming: pointer to NOR extended mode timing structure
	Return values	<ul style="list-style-type: none"> • HAL status
26.3.7	HAL_NOR_DeInit	
	Function Name	HAL_StatusTypeDef HAL_NOR_DeInit (NOR_HandleTypeDef * hnor)
	Function Description	Perform NOR memory De-Initialization sequence.
	Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
	Return values	<ul style="list-style-type: none"> • HAL status
26.3.8	HAL_NOR_MspInit	
	Function Name	void HAL_NOR_MspInit (NOR_HandleTypeDef * hnor)
	Function Description	NOR MSP Init.
	Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
	Return values	<ul style="list-style-type: none"> • None
26.3.9	HAL_NOR_MspDeInit	
	Function Name	void HAL_NOR_MspDeInit (NOR_HandleTypeDef * hnor)
	Function Description	NOR MSP DeInit.
	Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
	Return values	<ul style="list-style-type: none"> • None
26.3.10	HAL_NOR_MspWait	
	Function Name	void HAL_NOR_MspWait (NOR_HandleTypeDef * hnor, uint32_t Timeout)
	Function Description	NOR MSP Wait fro Ready/Busy signal.
	Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • Timeout: Maximum timeout value
	Return values	<ul style="list-style-type: none"> • None
26.3.11	HAL_NOR_Read_ID	

Function Name	HAL_StatusTypeDef HAL_NOR_Read_ID (NOR_HandleTypeDef * hnor, NOR_IDTypeDef * pNOR_ID)
Function Description	Read NOR flash IDs.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • pNOR_ID: : pointer to NOR ID structure
Return values	<ul style="list-style-type: none"> • HAL status

26.3.12 HAL_NOR_ReturnToReadMode

Function Name	HAL_StatusTypeDef HAL_NOR_ReturnToReadMode (NOR_HandleTypeDef * hnor)
Function Description	Returns the NOR memory to Read mode.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> • HAL status

26.3.13 HAL_NOR_Read

Function Name	HAL_StatusTypeDef HAL_NOR_Read (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)
Function Description	Read data from NOR memory.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • pAddress: pointer to Device address • pData: : pointer to read data
Return values	<ul style="list-style-type: none"> • HAL status

26.3.14 HAL_NOR_Program

Function Name	HAL_StatusTypeDef HAL_NOR_Program (NOR_HandleTypeDef * hnor, uint32_t * pAddress, uint16_t * pData)
Function Description	Program data to NOR memory.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • pAddress: Device address • pData: : pointer to the data to write
Return values	<ul style="list-style-type: none"> • HAL status

26.3.15 HAL_NOR_ReadBuffer

Function Name	HAL_StatusTypeDef HAL_NOR_ReadBuffer (NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)
Function Description	Reads a block of data from the FSMC NOR memory.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that

- contains the configuration information for NOR module.
- **uwAddress:** NOR memory internal address to read from.
- **pData:** pointer to the buffer that receives the data read from the NOR memory.
- **uwBufferSize:** : number of Half word to read.

Return values

- HAL status

26.3.16 HAL_NOR_ProgramBuffer

Function Name

HAL_StatusTypeDef HAL_NOR_ProgramBuffer
(NOR_HandleTypeDef * hnor, uint32_t uwAddress, uint16_t * pData, uint32_t uwBufferSize)

Function Description

Writes a half-word buffer to the FSMC NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **uwAddress:** NOR memory internal address from which the data
- **pData:** pointer to source data buffer.
- **uwBufferSize:** number of Half words to write.

Return values

- HAL status

Notes

- Some NOR memory need Address aligned to xx bytes (can be aligned to 64 bytes boundary for example).
- The maximum buffer size allowed is NOR memory dependent (can be 64 Bytes max for example).

26.3.17 HAL_NOR_Erase_Block

Function Name

HAL_StatusTypeDef HAL_NOR_Erase_Block
(NOR_HandleTypeDef * hnor, uint32_t BlockAddress, uint32_t Address)

Function Description

Erase the specified block of the NOR memory.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **BlockAddress:** : Block to erase address
- **Address:** Device address

Return values

- HAL status

26.3.18 HAL_NOR_Erase_Chip

Function Name

HAL_StatusTypeDef HAL_NOR_Erase_Chip
(NOR_HandleTypeDef * hnor, uint32_t Address)

Function Description

Erase the entire NOR chip.

Parameters

- **hnor:** pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
- **Address:** : Device address

Return values

- HAL status

26.3.19 HAL_NOR_Read_CFI

Function Name	HAL_StatusTypeDef HAL_NOR_Read_CFI (NOR_HandleTypeDef * hnor, NOR_CFITypeDef * pNOR_CFI)
Function Description	Read NOR flash CFI IDs.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • pNOR_CFI: : pointer to NOR CFI IDs structure
Return values	<ul style="list-style-type: none"> • HAL status

26.3.20 HAL_NOR_WriteOperation_Enable

Function Name	HAL_StatusTypeDef HAL_NOR_WriteOperation_Enable (NOR_HandleTypeDef * hnor)
Function Description	Enables dynamically NOR write operation.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> • HAL status

26.3.21 HAL_NOR_WriteOperation_Disable

Function Name	HAL_StatusTypeDef HAL_NOR_WriteOperation_Disable (NOR_HandleTypeDef * hnor)
Function Description	Disables dynamically NOR write operation.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> • HAL status

26.3.22 HAL_NOR_GetState

Function Name	HAL_NOR_StateTypeDef HAL_NOR_GetState (NOR_HandleTypeDef * hnor)
Function Description	return the NOR controller state
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module.
Return values	<ul style="list-style-type: none"> • NOR controller state

26.3.23 HAL_NOR_GetStatus

Function Name	HAL_NOR_StatusTypeDef HAL_NOR_GetStatus (NOR_HandleTypeDef * hnor, uint32_t Address, uint32_t Timeout)
Function Description	Returns the NOR operation status.
Parameters	<ul style="list-style-type: none"> • hnor: pointer to a NOR_HandleTypeDef structure that contains the configuration information for NOR module. • Address: Device address • Timeout: NOR programming Timeout
Return values	<ul style="list-style-type: none"> • NOR_Status The returned value can be: HAL_NOR_STATUS_SUCCESS,

HAL_NOR_STATUS_ERROR or
HAL_NOR_STATUS_TIMEOUT

26.4 NOR Firmware driver defines

26.4.1 NOR

NOR Exported Macros

`__HAL_NOR_RESET_HANDLE_STATE`

Description:

- Reset NOR handle state.

Parameters:

- `__HANDLE__`: NOR handle

Return value:

- None

NOR Private Constants

`NOR_CMD_ADDRESS_FIRST`

`NOR_CMD_ADDRESS_FIRST_CFI`

`NOR_CMD_ADDRESS_SECOND`

`NOR_CMD_ADDRESS_THIRD`

`NOR_CMD_ADDRESS_FOURTH`

`NOR_CMD_ADDRESS_FIFTH`

`NOR_CMD_ADDRESS_SIXTH`

`NOR_CMD_DATA_READ_RESET`

`NOR_CMD_DATA_FIRST`

`NOR_CMD_DATA_SECOND`

`NOR_CMD_DATA_AUTO_SELECT`

`NOR_CMD_DATA_PROGRAM`

`NOR_CMD_DATA_CHIP_BLOCK_ERASE_THIRD`

`NOR_CMD_DATA_CHIP_BLOCK_ERASE_FOURTH`

`NOR_CMD_DATA_CHIP_BLOCK_ERASE_FIFTH`

`NOR_CMD_DATA_CHIP_ERASE`

`NOR_CMD_DATA_CFI`

`NOR_CMD_DATA_BUFFER_AND_PROG`

`NOR_CMD_DATA_BUFFER_AND_PROG_CONFIRM`

`NOR_CMD_DATA_BLOCK_ERASE`

`NOR_MASK_STATUS_DQ5`

`NOR_MASK_STATUS_DQ6`

`MC_ADDRESS`

DEVICE_CODE1_ADDR
DEVICE_CODE2_ADDR
DEVICE_CODE3_ADDR
CFI1_ADDRESS
CFI2_ADDRESS
CFI3_ADDRESS
CFI4_ADDRESS
NOR_TMEOUT
NOR_MEMORY_8B
NOR_MEMORY_16B
NOR_MEMORY_ADDRESS1
NOR_MEMORY_ADDRESS2
NOR_MEMORY_ADDRESS3
NOR_MEMORY_ADDRESS4

NOR Private Macros

NOR_ADDR_SHIFT

Description:

- NOR memory address shifting.

Parameters:

- __NOR_ADDRESS: NOR base address
- __NOR_MEMORY_WIDTH__: NOR memory width
- __ADDRESS__: NOR memory address

Return value:

- NOR: shifted address value

NOR_WRITE

Description:

- NOR memory write data to specified address.

Parameters:

- __ADDRESS__: NOR memory address
- __DATA__: Data to write

Return value:

- None

27 HAL OPAMP Generic Driver

27.1 HAL OPAMP Generic Driver

27.2 OPAMP Firmware driver registers structures

27.2.1 OPAMP_InitTypeDef

Data Fields

- *uint32_t* **PowerSupplyRange**
- *uint32_t* **PowerMode**
- *uint32_t* **Mode**
- *uint32_t* **InvertingInput**
- *uint32_t* **NonInvertingInput**
- *uint32_t* **UserTrimming**
- *uint32_t* **TrimmingValueP**
- *uint32_t* **TrimmingValueN**
- *uint32_t* **TrimmingValuePLowPower**
- *uint32_t* **TrimmingValueNLowPower**

Field Documentation

- *uint32_t* **OPAMP_InitTypeDef::PowerSupplyRange**
Specifies the power supply range: above or under 2.4V. This parameter must be a value of [OPAMP_PowerSupplyRange](#) Caution: This parameter is common to all OPAMP instances: a modification of this parameter for the selected OPAMP impacts the other OPAMP instances.
- *uint32_t* **OPAMP_InitTypeDef::PowerMode**
Specifies the power mode Normal or Low-Power. This parameter must be a value of [OPAMP_PowerMode](#)
- *uint32_t* **OPAMP_InitTypeDef::Mode**
Specifies the OPAMP mode This parameter must be a value of [OPAMP_Mode](#) mode is either Standalone or Follower
- *uint32_t* **OPAMP_InitTypeDef::InvertingInput**
Specifies the inverting input in Standalone mode In Standalone mode: i.e when mode is OPAMP_STANDALONE_MODE This parameter must be a value of [OPAMP_InvertingInput](#) InvertingInput is either VM0 or VM1 In Follower mode: i.e when mode is OPAMP_FOLLOWER_MODE This parameter is Not Applicable
- *uint32_t* **OPAMP_InitTypeDef::NonInvertingInput**
Specifies the non inverting input of the opamp: This parameter must be a value of [OPAMP_NonInvertingInput](#) Note: Non-inverting input availability depends on OPAMP instance: OPAMP1: Non-inverting input is either IO0, DAC_Channel1 OPAMP2: Non-inverting input is either IO0, DAC_Channel1, DAC_Channel2 OPAMP3: Non-inverting input is either IO0, DAC_Channel2 (OPAMP3 availability depends on STM32L1 devices)
- *uint32_t* **OPAMP_InitTypeDef::UserTrimming**
Specifies the trimming mode This parameter must be a value of [OPAMP_UserTrimming](#) UserTrimming is either factory or user trimming. Caution:

This parameter is common to all OPAMP instances: a modification of this parameter for the selected OPAMP impacts the other OPAMP instances.

- ***uint32_t OPAMP_InitTypeDef::TrimmingValueP***
Specifies the offset trimming value (PMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 30 (Trimming value 31 is forbidden) 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueN***
Specifies the offset trimming value (NMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 30 (Trimming value 31 is forbidden) 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValuePLowPower***
Specifies the offset trimming value (PMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 30 (Trimming value 31 is forbidden) 16 is typical default value
- ***uint32_t OPAMP_InitTypeDef::TrimmingValueNLowPower***
Specifies the offset trimming value (NMOS) i.e. when UserTrimming is OPAMP_TRIMMING_USER. This parameter must be a number between Min_Data = 0 and Max_Data = 30 (Trimming value 31 is forbidden) 16 is typical default value

27.2.2 OPAMP_HandleTypeDef

Data Fields

- ***OPAMP_TypeDef * Instance***
- ***OPAMP_InitTypeDef Init***
- ***HAL_StatusTypeDef Status***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_OPAMP_StateTypeDef State***

Field Documentation

- ***OPAMP_TypeDef* OPAMP_HandleTypeDef::Instance***
OPAMP instance's registers base address
- ***OPAMP_InitTypeDef OPAMP_HandleTypeDef::Init***
OPAMP required parameters
- ***HAL_StatusTypeDef OPAMP_HandleTypeDef::Status***
OPAMP peripheral status
- ***HAL_LockTypeDef OPAMP_HandleTypeDef::Lock***
Locking object
- ***__IO HAL_OPAMP_StateTypeDef OPAMP_HandleTypeDef::State***
OPAMP communication state

27.3 OPAMP Firmware driver API description

27.3.1 OPAMP Peripheral Features

The device integrates up to 3 operational amplifiers OPAMP1, OPAMP2, OPAMP3 (OPAMP3 availability depends on device category)

1. The OPAMP(s) provides several exclusive running modes.
 - Standalone mode

- Follower mode
- 2. All OPAMP (same for all OPAMPs) can operate in
 - Either Low range ($V_{DDA} < 2.4V$) power supply
 - Or High range ($V_{DDA} > 2.4V$) power supply
- 3. Each OPAMP(s) can be configured in normal and low power mode.
- 4. The OPAMP(s) provide(s) calibration capabilities.
 - Calibration aims at correcting some offset for running mode.
 - The OPAMP uses either factory calibration settings OR user defined calibration (trimming) settings (i.e. trimming mode).
 - The user defined settings can be figured out using self calibration handled by HAL_OPAMP_SelfCalibrate, HAL_OPAMPEx_SelfCalibrateAll
 - HAL_OPAMP_SelfCalibrate:
 - Runs automatically the calibration in 2 steps: for transistors differential pair high (PMOS) or low (NMOS)
 - Enables the user trimming mode
 - Updates the init structure with trimming values with fresh calibration results. The user may store the calibration results for larger (ex monitoring the trimming as a function of temperature for instance)
 - For devices having several OPAMPs, HAL_OPAMPEx_SelfCalibrateAll runs calibration of all OPAMPs in parallel to save search time.
- 5. Running mode: Standalone mode
 - Gain is set externally (gain depends on external loads).
 - Follower mode also possible externally by connecting the inverting input to the output.
- 6. Running mode: Follower mode
 - No Inverting Input is connected.
 - The OPAMP(s) output(s) are internally connected to inverting input.
- 7. The OPAMPs inverting input can be selected among the list shown in [Table 19: "OPAMPs inverting/non-inverting inputs for STM32L1 devices"](#).
- 8. The OPAMPs non inverting input can be selected among the list shown in [Table 19: "OPAMPs inverting/non-inverting inputs for STM32L1 devices"](#).

Table 19: OPAMPs inverting/non-inverting inputs for STM32L1 devices

	HAL parameter name	OPAMP1	OPAMP2	OPAMP3 ⁽¹⁾
Inverting inputs ⁽²⁾	VM0 VM1	PA2 VINM pin ⁽³⁾	PA7 VINM pin	PC2 VINM pin
non-inverting inputs	VP0 DAC_CH1 ⁽⁴⁾ DAC_CH2	PA1 DAC_CH1 -	PA6 DAC_CH1 DAC_CH2	PC1 - DAC_CH2

Notes:

⁽¹⁾ OPAMP3 availability depends on device category.

⁽²⁾ NA in follower mode.

⁽³⁾ OPAMP input OPAMPx_VINM are dedicated OPAMP pins, their availability depends on device package.

⁽⁴⁾ DAC channels 1 and 2 are connected internally to OPAMP. Nevertheless, I/O pins connected to DAC can still be used as DAC output (pins PA4 and PA5).

Table 20: OPAMP outputs for STM32L1 devices

	OPAMP1	OPAMP2	OPAMP3 ⁽¹⁾
--	--------	--------	-----------------------

	OPAMP1	OPAMP2	OPAMP3 ⁽¹⁾
Output	PA3	PB0	PC3

Notes:

⁽¹⁾OPAMP3 availability depends on device category.

27.3.2 How to use this driver

power supply range

To run in low power mode:

1. Configure the opamp using HAL_OPAMP_Init() function:
 - Select OPAMP_POWER_SUPPLY_LOW (VDDA lower than 2.4V)
 - Otherwise select OPAMP_POWER_SUPPLY_HIGH (VDDA higher than 2.4V)

low / normal power mode

To run in low power mode:

1. Configure the opamp using HAL_OPAMP_Init() function:
 - Select OPAMP_POWERMODE_LOWPOWER
 - Otherwise select OPAMP_POWERMODE_NORMAL

Calibration

To run the opamp calibration self calibration:

1. Start calibration using HAL_OPAMP_SelfCalibrate. Store the calibration results.

Running mode

To use the opamp, perform the following steps:

1. Fill in the HAL_OPAMP_MspInit() to
 - Enable the OPAMP Peripheral clock using macro "`__HAL_RCC_OPAMP_CLK_ENABLE()`"
 - Configure the opamp input AND output in analog mode using HAL_GPIO_Init() to map the opamp output to the GPIO pin.
2. Configure the opamp using HAL_OPAMP_Init() function:
 - Select the mode
 - Select the inverting input
 - Select the non-inverting input
 - Select either factory or user defined trimming mode.
 - If the user defined trimming mode is enabled, select PMOS & NMOS trimming values (typ. settings returned by HAL_OPAMP_SelfCalibrate function).
3. Enable the opamp using HAL_OPAMP_Start() function.
4. Disable the opamp using HAL_OPAMP_Stop() function.
5. Lock the opamp in running mode using HAL_OPAMP_Lock() function. Caution: On STM32L1, HAL OPAMP lock is software lock only (not hardware lock as on some other STM32 devices)
6. If needed, unlock the opamp using HAL_OPAMPEX_Unlock() function.

Running mode: change of configuration while OPAMP ON

To Re-configure OPAMP when OPAMP is ON (change on the fly)

1. If needed, Fill in the HAL_OPAMP_MspInit()
 - This is the case for instance if you wish to use new OPAMP I/O
2. Configure the opamp using HAL_OPAMP_Init() function:
 - As in configure case, selects first the parameters you wish to modify.
3. Change from low power mode to normal power mode (& vice versa) requires first HAL_OPAMP_DeInit() (force OPAMP OFF) and then HAL_OPAMP_Init(). In other words, if OPAMP is ON, HAL_OPAMP_Init can NOT change power mode alone.

27.3.3 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- [HAL_OPAMP_Init\(\)](#)
- [HAL_OPAMP_DeInit\(\)](#)
- [HAL_OPAMP_MspInit\(\)](#)
- [HAL_OPAMP_MspDeInit\(\)](#)

27.3.4 IO operation functions

This subsection provides a set of functions allowing to manage the OPAMP start, stop and calibration actions.

This section contains the following APIs:

- [HAL_OPAMP_Start\(\)](#)
- [HAL_OPAMP_Stop\(\)](#)
- [HAL_OPAMP_SelfCalibrate\(\)](#)

27.3.5 Peripheral Control functions

This subsection provides a set of functions allowing to control the OPAMP data transfers.

This section contains the following APIs:

- [HAL_OPAMP_Lock\(\)](#)
- [HAL_OPAMP_GetTrimOffset\(\)](#)

27.3.6 Peripheral State functions

This subsection permit to get in run-time the status of the peripheral.

This section contains the following APIs:

- [HAL_OPAMP_GetState\(\)](#)

27.3.7 HAL_OPAMP_Init

Function Name	HAL_StatusTypeDef HAL_OPAMP_Init (OPAMP_HandleTypeDef * hopamp)
Function Description	Initializes the OPAMP according to the specified parameters in the OPAMP_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hopamp: OPAMP handle
Return values	<ul style="list-style-type: none"> • HAL status

Notes

- If the selected opamp is locked, initialization can't be performed. To unlock the configuration, perform a system reset.

27.3.8 HAL_OPAMP_DeInit

Function Name **HAL_StatusTypeDef HAL_OPAMP_DeInit (OPAMP_HandleTypeDef * hopamp)**

Function Description DeInitializes the OPAMP peripheral.

Parameters • **hopamp:** OPAMP handle

Return values • HAL status

Notes • Deinitialization can be performed if the OPAMP configuration is locked. (the OPAMP lock is SW in STM32L1)

27.3.9 HAL_OPAMP_MspInit

Function Name **void HAL_OPAMP_MspInit (OPAMP_HandleTypeDef * hopamp)**

Function Description Initializes the OPAMP MSP.

Parameters • **hopamp:** OPAMP handle

Return values • None

27.3.10 HAL_OPAMP_MspDeInit

Function Name **void HAL_OPAMP_MspDeInit (OPAMP_HandleTypeDef * hopamp)**

Function Description DeInitializes OPAMP MSP.

Parameters • **hopamp:** OPAMP handle

Return values • None

27.3.11 HAL_OPAMP_Start

Function Name **HAL_StatusTypeDef HAL_OPAMP_Start (OPAMP_HandleTypeDef * hopamp)**

Function Description Start the opamp.

Parameters • **hopamp:** OPAMP handle

Return values • HAL status

27.3.12 HAL_OPAMP_Stop

Function Name **HAL_StatusTypeDef HAL_OPAMP_Stop (OPAMP_HandleTypeDef * hopamp)**

Function Description Stop the opamp.

Parameters • **hopamp:** OPAMP handle

Return values • HAL status

27.3.13 HAL_OPAMP_SelfCalibrate

Function Name	HAL_StatusTypeDef HAL_OPAMP_SelfCalibrate (OPAMP_HandleTypeDef * hopamp)
Function Description	Run the self calibration of one OPAMP.
Parameters	<ul style="list-style-type: none"> hopamp: handle
Return values	<ul style="list-style-type: none"> Updated offset trimming values (PMOS & NMOS), user trimming is enabled HAL status
Notes	<ul style="list-style-type: none"> Trimming values (PMOS & NMOS) are updated and user trimming is enabled if calibration is succesful. Calibration is performed in the mode specified in OPAMP init structure (mode normal or low-power). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated. Calibration runs about 10 ms.

27.3.14 HAL_OPAMP_Lock

Function Name	HAL_StatusTypeDef HAL_OPAMP_Lock (OPAMP_HandleTypeDef * hopamp)
Function Description	Lock the selected opamp configuration.
Parameters	<ul style="list-style-type: none"> hopamp: OPAMP handle
Return values	<ul style="list-style-type: none"> HAL status

27.3.15 HAL_OPAMP_GetTrimOffset

Function Name	HAL_OPAMP_TrimmingValueTypeDef HAL_OPAMP_GetTrimOffset (OPAMP_HandleTypeDef * hopamp, uint32_t trimmingoffset)
Function Description	Return the OPAMP factory trimming value Caution: On STM32L1 OPAMP, user can retrieve factory trimming if OPAMP has never been set to user trimming before.
Parameters	<ul style="list-style-type: none"> hopamp: : OPAMP handle trimmingoffset: : Trimming offset (P or N) This parameter must be a value of OPAMP FactoryTrimming
Return values	<ul style="list-style-type: none"> Trimming value (P or N): range: 0->31 or OPAMP_FACTORYTRIMMING_DUMMY if trimming value is not available
Notes	<ul style="list-style-type: none"> Calibration parameter retrieved is corresponding to the mode specified in OPAMP init structure (mode normal or low-power). To retrieve calibration parameters for both modes, repeat this function after OPAMP init structure accordingly updated.

27.3.16 HAL_OPAMP_GetState

Function Name	HAL_OPAMP_StateTypeDef HAL_OPAMP_GetState (OPAMP_HandleTypeDef * hopamp)
---------------	---

Function Description	Return the OPAMP state.
Parameters	<ul style="list-style-type: none"> hopamp: : OPAMP handle
Return values	<ul style="list-style-type: none"> HAL state

27.4 OPAMP Firmware driver defines

27.4.1 OPAMP

OPAMP Exported Constants

OPAMP_TRIM_VALUE_MASK

OPAMP_CSR_INSTANCE_OFFSET

OPAMP_OTR_INSTANCE_OFFSET

OPAMP FactoryTrimming

OPAMP_FACTORYTRIMMING_DUMMY Dummy value if trimming value could not be retrieved

OPAMP_FACTORYTRIMMING_P Offset trimming P

OPAMP_FACTORYTRIMMING_N Offset trimming N

OPAMP InvertingInput

OPAMP_INVERTINGINPUT_IO0 Comparator inverting input connected to dedicated IO pin low-leakage

OPAMP_INVERTINGINPUT_IO1 Comparator inverting input connected to alternative IO pin available on some device packages

OPAMP Mode

OPAMP_STANDALONE_MODE OPAMP standalone mode

OPAMP_FOLLOWER_MODE OPAMP follower mode

OPAMP NonInvertingInput

OPAMP_NONINVERTINGINPUT_IO0 Comparator non-inverting input connected to dedicated IO pin low-leakage

OPAMP_NONINVERTINGINPUT_DAC_CH1 Comparator non-inverting input connected internally to DAC channel 1. Available only on OPAMP1 and OPAMP2.

OPAMP_NONINVERTINGINPUT_DAC_CH2 Comparator non-inverting input connected internally to DAC channel 2. Available only on OPAMP2 and OPAMP3 (OPAMP3 availability depends on STM32L1 devices).

OPAMP PowerMode

OPAMP_POWERMODE_NORMAL

OPAMP_POWERMODE_LOWPOWER

OPAMP PowerSupplyRange

OPAMP_POWERSUPPLY_LOW Power supply range low (VDDA lower than 2.4V)

OPAMP_POWERSUPPLY_HIGH Power supply range high (VDDA higher than 2.4V)

OPAMP Private Constants

OPAMP_TRIMMING_DELAY

OPAMP Private Macro

__HAL_OPAMP_RESET_HANDLE_STATE

Description:

- Reset OPAMP handle state.

Parameters:

- __HANDLE__: OPAMP handle.

Return value:

- None

OPAMP_CSR_OPAXPD

Description:

- Select the OPAMP bit OPAXPD (power-down) corresponding to the selected OPAMP instance.

Parameters:

- __HANDLE__: OPAMP handle

Return value:

- None

OPAMP_CSR_S3SELX

Description:

- Select the OPAMP bit S3SELx (switch 3) corresponding to the selected OPAMP instance.

Parameters:

- __HANDLE__: OPAMP handle

Return value:

- None

OPAMP_CSR_S4SELX

Description:

- Select the OPAMP bit S4SELx (switch 4) corresponding to the selected OPAMP instance.

Parameters:

- __HANDLE__: OPAMP handle

Return value:

- None

OPAMP_CSR_S5SELX

Description:

- Select the OPAMP bit S5SELx (switch 5) corresponding to the selected OPAMP instance.

Parameters:

- __HANDLE__: OPAMP handle

OPAMP_CSR_S6SELX

Return value:

- None

Description:

- Select the OPAMP bit S3SELx (switch 6) corresponding to the selected OPAMP instance.

Parameters:

- __HANDLE__: OPAMP handle

Return value:

- None

OPAMP_CSR_OPAXCAL_L

Description:

- Select the OPAMP bit OPAXCAL_L (offset calibration for differential pair P) corresponding to the selected OPAMP instance.

Parameters:

- __HANDLE__: OPAMP handle

Return value:

- None

OPAMP_CSR_OPAXCAL_H

Description:

- Select the OPAMP bit OPAXCAL_H (offset calibration for differential pair N) corresponding to the selected OPAMP instance.

Parameters:

- __HANDLE__: OPAMP handle

Return value:

- None

OPAMP_CSR_OPAXLPM

Description:

- Select the OPAMP bit OPAXLPM (low power mode) corresponding to the selected OPAMP instance.

Parameters:

- __HANDLE__: OPAMP handle

Return value:

- None

OPAMP_CSR_ALL_SWITCHES

Description:

- Select the OPAMP bits of all switches corresponding to the selected OPAMP instance.

OPAMP_CSR_ANAWSELX

Parameters:

- `__HANDLE__`: OPAMP handle

Return value:

- None

Description:

- Select the OPAMP bit ANAWSELx (switch SanA) corresponding to the selected OPAMP instance.

Parameters:

- `__HANDLE__`: OPAMP handle

Return value:

- None

Description:

- Select the OPAMP bit OPAXCALOUT in function of the selected OPAMP instance.

Parameters:

- `__HANDLE__`: OPAMP handle

Return value:

- None

OPAMP_OFFSET_TRIM_BITSPPOSITION

Description:

- Select the OPAMP trimming bits position value (position of LSB) in register OPAMP_OTR or register OPAMP_LPOTR in function of the selected OPAMP instance and the transistors differential pair high (PMOS) or low (NMOS).

Parameters:

- `__HANDLE__`: OPAMP handle
- `__TRIM_HIGH_LOW__`: transistors differential pair high or low. Must be a value of

Return value:

- None

OPAMP_OFFSET_TRIM_SET

Description:

- Shift the OPAMP trimming bits to register OPAMP_OTR or register OPAMP_LPOTR in function of the selected OPAMP instance and the transistors differential pair high (PMOS) or low (NMOS).

IS_OPAMP_TRIMMINGVALUE

Parameters:

- __HANDLE__: OPAMP handle
- __TRIM_HIGH_LOW__: transistors differential pair high or low. Must be a value of
- __TRIMMING_VALUE__: Trimming value

Return value:

- None

Description:

- Check that trimming value is within correct range.

Parameters:

- TRIMMINGVALUE: OPAMP trimming value

Return value:

- None

IS_OPAMP_FUNCTIONAL_NORMALMODE

IS_OPAMP_INVERTING_INPUT

IS_OPAMP_POWERMODE

IS_OPAMP_POWER_SUPPLY_RANGE

IS_OPAMP_TRIMMING

IS_OPAMP_FACTORYTRIMMING

OPAMP User Trimming

OPAMP_TRIMMING_FACTORY Factory trimming

OPAMP_TRIMMING_USER User trimming

28 HAL OPAMP Extension Driver

28.1 HAL OPAMP Extension Driver

28.2 OPAMPEx Firmware driver API description

28.2.1 Peripheral Control functions

- OPAMP unlock.

This section contains the following APIs:

- [*HAL_OPAMPEx_Unlock\(\)*](#)

28.2.2 Extended IO operation functions

- OPAMP Self calibration.

This section contains the following APIs:

- [*HAL_OPAMPEx_SelfCalibrateAll\(\)*](#)

28.2.3 HAL_OPAMPEx_Unlock

Function Name	HAL_StatusTypeDef HAL_OPAMPEx_Unlock (OPAMP_HandleTypeDef * hopamp)
Function Description	Unlock the selected opamp configuration.
Parameters	<ul style="list-style-type: none"> • hopamp: OPAMP handle
Return values	<ul style="list-style-type: none"> • HAL status

28.2.4 HAL_OPAMPEx_SelfCalibrateAll

Function Name	HAL_StatusTypeDef HAL_OPAMPEx_SelfCalibrateAll (OPAMP_HandleTypeDef * hopamp1, OPAMP_HandleTypeDef * hopamp2, OPAMP_HandleTypeDef * hopamp3)
Function Description	Run the self calibration of the 3 OPAMPs in parallel.
Parameters	<ul style="list-style-type: none"> • hopamp1: handle • hopamp2: handle • hopamp3: handle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • Trimming values (PMOS & NMOS) are updated and user trimming is enabled is calibration is succesful. • Calibration is performed in the mode specified in OPAMP init structure (mode normal or low-power). To perform calibration for both modes, repeat this function twice after OPAMP init structure accordingly updated. • Calibration runs about 10 ms (5 dichotmy steps, repeated for

P and N transistors: 10 steps with 1 ms for each step).

28.3 OPAMPEX Firmware driver defines

28.3.1 OPAMPEX

OPAMPEX Exported Constants

OPAMP_CSR_OPAXPD_ALL

OPAMP_CSR_OPAXCAL_L_ALL

OPAMP_CSR_OPAXCAL_H_ALL

OPAMP_CSR_ALL_SWITCHES_ALL_OPAMPS

OPAMPEX Exported Macro

__HAL_OPAMP_OPAMP3OUT_CONNECT_ADC_COMP1

Description:

- Enable internal analog switch SW1 to connect OPAMP3 output to ADC switch matrix (ADC channel VCOMP, channel 26) and COMP1 non-inverting input (OPAMP3 available on STM32L1 devices Cat.4 only).

Return value:

- None

__HAL_OPAMP_OPAMP3OUT_DISCONNECT_ADC_COMP1

Description:

- Disable internal analog switch SW1 to disconnect OPAMP3 output from ADC switch matrix (ADC channel VCOMP, channel 26) and COMP1 non-inverting input.

Return value:

- None

OPAMPEX Private Macro

OPAMP_INSTANCE_DECIMAL

Description:

- Get the OPAMP instance in decimal number for further processing needs by

HAL OPAMP driver functions.

Parameters:

- `__HANDLE__`: OPAMP handle

Return value:

- 0: for OPAMP1, "1" for OPAMP2, "2" for OPAMP3

IS_OPAMP_NONINVERTING_INPUT_CHECK_INSTANCE**Description:**

- Check OPAMP non-inverting input in function of OPAMPx instance.

Parameters:

- `__HANDLE__`: OPAMP handle
- `INPUT`: OPAMP non-inverting input

Return value:

- None

29 HAL PCD Generic Driver

29.1 HAL PCD Generic Driver

29.2 PCD Firmware driver registers structures

29.2.1 PCD_InitTypeDef

Data Fields

- *uint32_t dev_endpoints*
- *uint32_t speed*
- *uint32_t ep0_mps*
- *uint32_t phy_iface*
- *uint32_t Sof_enable*
- *uint32_t low_power_enable*
- *uint32_t lpm_enable*
- *uint32_t battery_charging_enable*

Field Documentation

- *uint32_t PCD_InitTypeDef::dev_endpoints*
Device Endpoints number. This parameter depends on the used USB core. This parameter must be a number between Min_Data = 1 and Max_Data = 15
- *uint32_t PCD_InitTypeDef::speed*
USB Core speed. This parameter can be any value of [PCD_Core_Speed](#)
- *uint32_t PCD_InitTypeDef::ep0_mps*
Set the Endpoint 0 Max Packet size. This parameter can be any value of [PCD_EP0_MPS](#)
- *uint32_t PCD_InitTypeDef::phy_iface*
Select the used PHY interface. This parameter can be any value of [PCD_Core_PHY](#)
- *uint32_t PCD_InitTypeDef::Sof_enable*
Enable or disable the output of the SOF signal.
- *uint32_t PCD_InitTypeDef::low_power_enable*
Enable or disable Low Power mode
- *uint32_t PCD_InitTypeDef::lpm_enable*
Enable or disable Battery charging.
- *uint32_t PCD_InitTypeDef::battery_charging_enable*
Enable or disable Battery charging.

29.2.2 PCD_EPTTypeDef

Data Fields

- *uint8_t num*
- *uint8_t is_in*
- *uint8_t is_stall*

- ***uint8_t type***
- ***uint16_t pmaaddress***
- ***uint16_t pmaaddr0***
- ***uint16_t pmaaddr1***
- ***uint8_t doublebuffer***
- ***uint32_t maxpacket***
- ***uint8_t * xfer_buff***
- ***uint32_t xfer_len***
- ***uint32_t xfer_count***

Field Documentation

- ***uint8_t PCD_EPTypedef::num***
Endpoint number This parameter must be a number between Min_Data = 1 and Max_Data = 15
- ***uint8_t PCD_EPTypedef::is_in***
Endpoint direction This parameter must be a number between Min_Data = 0 and Max_Data = 1
- ***uint8_t PCD_EPTypedef::is_stall***
Endpoint stall condition This parameter must be a number between Min_Data = 0 and Max_Data = 1
- ***uint8_t PCD_EPTypedef::type***
Endpoint type This parameter can be any value of [PCD_EP_Type](#)
- ***uint16_t PCD_EPTypedef::pmaaddress***
PMA Address This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- ***uint16_t PCD_EPTypedef::pmaaddr0***
PMA Address0 This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- ***uint16_t PCD_EPTypedef::pmaaddr1***
PMA Address1 This parameter can be any value between Min_addr = 0 and Max_addr = 1K
- ***uint8_t PCD_EPTypedef::doublebuffer***
Double buffer enable This parameter can be 0 or 1
- ***uint32_t PCD_EPTypedef::maxpacket***
Endpoint Max packet size This parameter must be a number between Min_Data = 0 and Max_Data = 64KB
- ***uint8_t* PCD_EPTypedef::xfer_buff***
Pointer to transfer buffer
- ***uint32_t PCD_EPTypedef::xfer_len***
Current transfer length
- ***uint32_t PCD_EPTypedef::xfer_count***
Partial transfer length in case of multi packet transfer

29.2.3 PCD_HandleTypeDef

Data Fields

- ***PCD_TypeDef * Instance***
- ***PCD_InitTypeDef Init***
- ***__IO uint8_t USB_Address***

- ***PCD_EPTTypeDef IN_ep***
- ***PCD_EPTTypeDef OUT_ep***
- ***HAL_LockTypeDef Lock***
- ***__IO PCD_StateTypeDef State***
- ***uint32_t Setup***
- ***void *pData***

Field Documentation

- ***PCD_TypeDef* PCD_HandleTypeDef::Instance***
Register base address
- ***PCD_InitTypeDef PCD_HandleTypeDef::Init***
PCD required parameters
- ***__IO uint8_t PCD_HandleTypeDef::USB_Address***
USB Address
- ***PCD_EPTTypeDef PCD_HandleTypeDef::IN_ep[8]***
IN endpoint parameters
- ***PCD_EPTTypeDef PCD_HandleTypeDef::OUT_ep[8]***
OUT endpoint parameters
- ***HAL_LockTypeDef PCD_HandleTypeDef::Lock***
PCD peripheral status
- ***__IO PCD_StateTypeDef PCD_HandleTypeDef::State***
PCD communication state
- ***uint32_t PCD_HandleTypeDef::Setup[12]***
Setup packet buffer
- ***void* PCD_HandleTypeDef::pData***
Pointer to upper stack Handler

29.3 PCD Firmware driver API description

29.3.1 How to use this driver

The PCD HAL driver can be used as follows:

1. Declare a PCD_HandleTypeDef handle structure, for example: PCD_HandleTypeDef hpcd;
2. Fill parameters of Init structure in HCD handle
3. Call HAL_PCD_Init() API to initialize the HCD peripheral (Core, Device core, ...)
4. Initialize the PCD low level resources through the HAL_PCD_MspInit() API:
 - a. Enable the PCD/USB Low Level interface clock using
– `__HAL_RCC_USB_CLK_ENABLE();`
 - b. Initialize the related GPIO clocks
 - c. Configure PCD pin-out
 - d. Configure PCD NVIC interrupt
5. Associate the Upper USB device stack to the HAL PCD Driver:
 - a. `hpcd.pData = pdev;`
6. Enable HCD transmission and reception:
 - a. `HAL_PCD_Start();`

29.3.2 Initialization and de-initialization functions

This section provides functions allowing to:

This section contains the following APIs:

- [*HAL_PCD_Init\(\)*](#)
- [*HAL_PCD_DeInit\(\)*](#)
- [*HAL_PCD_MspInit\(\)*](#)
- [*HAL_PCD_MspDeInit\(\)*](#)

29.3.3 IO operation functions

This subsection provides a set of functions allowing to manage the PCD data transfers.

This section contains the following APIs:

- [*HAL_PCD_Start\(\)*](#)
- [*HAL_PCD_Stop\(\)*](#)
- [*HAL_PCD_IRQHandler\(\)*](#)
- [*HAL_PCD_DataOutStageCallback\(\)*](#)
- [*HAL_PCD_DataInStageCallback\(\)*](#)
- [*HAL_PCD_SetupStageCallback\(\)*](#)
- [*HAL_PCD_SOFCallback\(\)*](#)
- [*HAL_PCD_ResetCallback\(\)*](#)
- [*HAL_PCD_SuspendCallback\(\)*](#)
- [*HAL_PCD_ResumeCallback\(\)*](#)
- [*HAL_PCD_ISOOUTIncompleteCallback\(\)*](#)
- [*HAL_PCD_ISOINIncompleteCallback\(\)*](#)
- [*HAL_PCD_ConnectCallback\(\)*](#)
- [*HAL_PCD_DisconnectCallback\(\)*](#)

29.3.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the PCD data transfers.

This section contains the following APIs:

- [*HAL_PCD_DevConnect\(\)*](#)
- [*HAL_PCD_DevDisconnect\(\)*](#)
- [*HAL_PCD_SetAddress\(\)*](#)
- [*HAL_PCD_EP_Open\(\)*](#)
- [*HAL_PCD_EP_Close\(\)*](#)
- [*HAL_PCD_EP_Receive\(\)*](#)
- [*HAL_PCD_EP_GetRxCount\(\)*](#)
- [*HAL_PCD_EP_Transmit\(\)*](#)
- [*HAL_PCD_EP_SetStall\(\)*](#)
- [*HAL_PCD_EP_ClrStall\(\)*](#)
- [*HAL_PCD_EP_Flush\(\)*](#)
- [*HAL_PCD_ActivateRemoteWakeup\(\)*](#)
- [*HAL_PCD_DeActivateRemoteWakeup\(\)*](#)

29.3.5 Peripheral State functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- [*HAL_PCD_GetState\(\)*](#)
- [*HAL_PCDEx_SetConnectionState\(\)*](#)

29.3.6 HAL_PCD_Init

Function Name **HAL_StatusTypeDef HAL_PCD_Init (PCD_HandleTypeDef ***

hpcd)

Function Description Initializes the PCD according to the specified parameters in the PCD_InitTypeDef and create the associated handle.

Parameters • **hpcd**: PCD handle

Return values • HAL status

29.3.7 HAL_PCD_DeInit

Function Name **HAL_StatusTypeDef HAL_PCD_DeInit (PCD_HandleTypeDef * hpcd)**

Function Description DeInitializes the PCD peripheral.

Parameters • **hpcd**: PCD handle

Return values • HAL status

29.3.8 HAL_PCD_MspInit

Function Name **void HAL_PCD_MspInit (PCD_HandleTypeDef * hpcd)**

Function Description Initializes the PCD MSP.

Parameters • **hpcd**: PCD handle

Return values • None

29.3.9 HAL_PCD_MspDeInit

Function Name **void HAL_PCD_MspDeInit (PCD_HandleTypeDef * hpcd)**

Function Description DeInitializes PCD MSP.

Parameters • **hpcd**: PCD handle

Return values • None

29.3.10 HAL_PCD_Start

Function Name **HAL_StatusTypeDef HAL_PCD_Start (PCD_HandleTypeDef * hpcd)**

Function Description Start The USB OTG Device.

Parameters • **hpcd**: PCD handle

Return values • HAL status

29.3.11 HAL_PCD_Stop

Function Name **HAL_StatusTypeDef HAL_PCD_Stop (PCD_HandleTypeDef * hpcd)**

Function Description Stop The USB OTG Device.

Parameters • **hpcd**: PCD handle

Return values • HAL status

29.3.12 HAL_PCD_IRQHandler

Function Name	void HAL_PCD_IRQHandler (PCD_HandleTypeDef * hpcd)
Function Description	This function handles PCD interrupt request.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• HAL status

29.3.13 HAL_PCD_DataOutStageCallback

Function Name	void HAL_PCD_DataOutStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)
Function Description	Data out stage callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• epnum: endpoint number
Return values	<ul style="list-style-type: none">• None

29.3.14 HAL_PCD_DataInStageCallback

Function Name	void HAL_PCD_DataInStageCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)
Function Description	Data IN stage callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle• epnum: endpoint number
Return values	<ul style="list-style-type: none">• None

29.3.15 HAL_PCD_SetupStageCallback

Function Name	void HAL_PCD_SetupStageCallback (PCD_HandleTypeDef * hpcd)
Function Description	Setup stage callback.
Parameters	<ul style="list-style-type: none">• hpcd: ppp handle
Return values	<ul style="list-style-type: none">• None

29.3.16 HAL_PCD_SOFCallback

Function Name	void HAL_PCD_SOFCallback (PCD_HandleTypeDef * hpcd)
Function Description	USB Start Of Frame callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle
Return values	<ul style="list-style-type: none">• None

29.3.17 HAL_PCD_ResetCallback

Function Name	void HAL_PCD_ResetCallback (PCD_HandleTypeDef * hpcd)
Function Description	USB Reset callbacks.
Parameters	<ul style="list-style-type: none">• hpcd: PCD handle

Return values

- None

29.3.18 HAL_PCD_SuspendCallback

Function Name **void HAL_PCD_SuspendCallback (PCD_HandleTypeDef * hpcd)**

Function Description Suspend event callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

29.3.19 HAL_PCD_ResumeCallback

Function Name **void HAL_PCD_ResumeCallback (PCD_HandleTypeDef * hpcd)**

Function Description Resume event callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

29.3.20 HAL_PCD_ISOOUTIncompleteCallback

Function Name **void HAL_PCD_ISOOUTIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**

Function Description Incomplete ISO OUT callbacks.

Parameters

- **hpcd**: PCD handle
- **epnum**: endpoint number

Return values

- None

29.3.21 HAL_PCD_ISOINIncompleteCallback

Function Name **void HAL_PCD_ISOINIncompleteCallback (PCD_HandleTypeDef * hpcd, uint8_t epnum)**

Function Description Incomplete ISO IN callbacks.

Parameters

- **hpcd**: PCD handle
- **epnum**: endpoint number

Return values

- None

29.3.22 HAL_PCD_ConnectCallback

Function Name **void HAL_PCD_ConnectCallback (PCD_HandleTypeDef * hpcd)**

Function Description Connection event callbacks.

Parameters

- **hpcd**: PCD handle

Return values

- None

29.3.23 HAL_PCD_DisconnectCallback

Function Name	void HAL_PCD_DisconnectCallback (PCD_HandleTypeDef * hpcd)
Function Description	Disconnection event callbacks.
Parameters	<ul style="list-style-type: none"> • hpcd: ppp handle
Return values	<ul style="list-style-type: none"> • None

29.3.24 HAL_PCD_DevConnect

Function Name	HAL_StatusTypeDef HAL_PCD_DevConnect (PCD_HandleTypeDef * hpcd)
Function Description	Connect the USB device.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • HAL status

29.3.25 HAL_PCD_DevDisconnect

Function Name	HAL_StatusTypeDef HAL_PCD_DevDisconnect (PCD_HandleTypeDef * hpcd)
Function Description	Disconnect the USB device.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle
Return values	<ul style="list-style-type: none"> • HAL status

29.3.26 HAL_PCD_SetAddress

Function Name	HAL_StatusTypeDef HAL_PCD_SetAddress (PCD_HandleTypeDef * hpcd, uint8_t address)
Function Description	Set the USB Device address.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • address: new device address
Return values	<ul style="list-style-type: none"> • HAL status

29.3.27 HAL_PCD_EP_Open

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Open (PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint16_t ep_mps, uint8_t ep_type)
Function Description	Open and configure an endpoint.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • ep_addr: endpoint address • ep_mps: endpoint max packert size • ep_type: endpoint type
Return values	<ul style="list-style-type: none"> • HAL status

29.3.28 HAL_PCD_EP_Close

Function Name	HAL_StatusTypeDef HAL_PCD_EP_Close
---------------	---

(PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function Description Deactivate an endpoint.

Parameters

- **hpcd**: PCD handle
- **ep_addr**: endpoint address

Return values

- HAL status

29.3.29 HAL_PCD_EP_Receive

Function Name **HAL_StatusTypeDef HAL_PCD_EP_Receive**
(PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf,
uint32_t len)

Function Description Receive an amount of data.

Parameters

- **hpcd**: PCD handle
- **ep_addr**: endpoint address
- **pBuf**: pointer to the reception buffer
- **len**: amount of data to be received

Return values

- HAL status

29.3.30 HAL_PCD_EP_GetRxCount

Function Name **uint16_t HAL_PCD_EP_GetRxCount** (PCD_HandleTypeDef *
hpcd, uint8_t ep_addr)

Function Description Get Received Data Size.

Parameters

- **hpcd**: PCD handle
- **ep_addr**: endpoint address

Return values

- Data Size

29.3.31 HAL_PCD_EP_Transmit

Function Name **HAL_StatusTypeDef HAL_PCD_EP_Transmit**
(PCD_HandleTypeDef * hpcd, uint8_t ep_addr, uint8_t * pBuf,
uint32_t len)

Function Description Send an amount of data.

Parameters

- **hpcd**: PCD handle
- **ep_addr**: endpoint address
- **pBuf**: pointer to the transmission buffer
- **len**: amount of data to be sent

Return values

- HAL status

29.3.32 HAL_PCD_EP_SetStall

Function Name **HAL_StatusTypeDef HAL_PCD_EP_SetStall**
(PCD_HandleTypeDef * hpcd, uint8_t ep_addr)

Function Description Set a STALL condition over an endpoint.

Parameters

- **hpcd**: PCD handle
- **ep_addr**: endpoint address

Return values

- HAL status

29.3.33 HAL_PCD_EP_ClrStall

Function Name **HAL_StatusTypeDef HAL_PCD_EP_ClrStall (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**

Function Description Clear a STALL condition over in an endpoint.

Parameters

- **hpcd**: PCD handle
- **ep_addr**: endpoint address

Return values

- HAL status

29.3.34 HAL_PCD_EP_Flush

Function Name **HAL_StatusTypeDef HAL_PCD_EP_Flush (PCD_HandleTypeDef * hpcd, uint8_t ep_addr)**

Function Description Flush an endpoint.

Parameters

- **hpcd**: PCD handle
- **ep_addr**: endpoint address

Return values

- HAL status

29.3.35 HAL_PCD_ActivateRemoteWakeup

Function Name **HAL_StatusTypeDef HAL_PCD_ActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)**

Function Description HAL_PCD_ActivateRemoteWakeup : active remote wakeup signalling.

Parameters

- **hpcd**: PCD handle

Return values

- status

29.3.36 HAL_PCD_DeActivateRemoteWakeup

Function Name **HAL_StatusTypeDef HAL_PCD_DeActivateRemoteWakeup (PCD_HandleTypeDef * hpcd)**

Function Description HAL_PCD_DeActivateRemoteWakeup : de-active remote wakeup signalling.

Parameters

- **hpcd**: PCD handle

Return values

- status

29.3.37 HAL_PCD_GetState

Function Name **PCD_StateTypeDef HAL_PCD_GetState (PCD_HandleTypeDef * hpcd)**

Function Description Return the PCD state.

Parameters

- **hpcd**: : PCD handle

Return values

- HAL state

29.3.38 HAL_PCDEx_SetConnectionState

Function Name	void HAL_PCDEx_SetConnectionState (PCD_HandleTypeDef * hpcd, uint8_t state)
Function Description	Software Device Connection.
Parameters	<ul style="list-style-type: none"> • hpcd: PCD handle • state: Device state
Return values	<ul style="list-style-type: none"> • None

29.4 PCD Firmware driver defines**29.4.1 PCD*****PCD Core PHY***

PCD_PHY_EMBEDDED

PCD Core Speed

PCD_SPEED_HIGH

PCD_SPEED_FULL

PCD_ENDP_Type

PCD_ENDP0

PCD_ENDP1

PCD_ENDP2

PCD_ENDP3

PCD_ENDP4

PCD_ENDP5

PCD_ENDP6

PCD_ENDP7

PCD_SNG_BUF

PCD_DBL_BUF

IS_PCD_ALL_INSTANCE

PCD EP0 MPS

DEP0CTL_MPS_64

DEP0CTL_MPS_32

DEP0CTL_MPS_16

DEP0CTL_MPS_8

PCD_EP0MPS_64

PCD_EP0MPS_32

PCD_EP0MPS_16

PCD_EP0MPS_08

PCD EP Type

PCD_EP_TYPE_CTRL

PCD_EP_TYPE_ISOC

PCD_EP_TYPE_BULK

PCD_EP_TYPE_INTR

PCD Exported Macros

__HAL_PCD_GET_FLAG

__HAL_PCD_CLEAR_FLAG

__HAL_USB_WAKEUP_EXTI_ENABLE_IT

__HAL_USB_WAKEUP_EXTI_DISABLE_IT

__HAL_USB_WAKEUP_EXTI_GET_FLAG

__HAL_USB_WAKEUP_EXTI_CLEAR_FLAG

__HAL_USB_WAKEUP_EXTI_ENABLE_RISING_EDGE

__HAL_USB_WAKEUP_EXTI_ENABLE_FALLING_EDGE

__HAL_USB_WAKEUP_EXTI_ENABLE_RISING_FALLING_EDGE

PCD Exti Line Wakeup

USB_WAKEUP_EXTI_LINE External interrupt line 18 Connected to the USB FS EXTI Line

PCD Private Constants

BTABLE_ADDRESS

PCD Private Macros

PCD_SET_ENDPOINT

PCD_GET_ENDPOINT

PCD_SET_EPTYPE

Description:

- sets the type in the endpoint register(bits EP_TYPE[1:0])

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wType: Endpoint Type.

Return value:

- None

PCD_GET_EPTYPE

Description:

- gets the type in the endpoint register(bits EP_TYPE[1:0])

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

PCD_FreeUserBuffer**Return value:**

- Endpoint: Type

Description:

- free buffer used from the application realizing it to the line toggles bit SW_BUF in the double buffered endpoint register

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- bDir: Direction

Return value:

- None

PCD_GET_DB_DIR**Description:**

- gets direction of the double buffered endpoint

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- EP_DBUF_OUT: if the endpoint counter not yet programmed.

PCD_SET_EP_TX_STATUS**Description:**

- sets the status for tx transfer (bits STAT_TX[1:0]).

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wState: new state

Return value:

- None

PCD_SET_EP_RX_STATUS**Description:**

- sets the status for rx transfer (bits STAT_TX[1:0])

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wState: new state

Return value:

PCD_SET_EP_TXRX_STATUS

- None

Description:

- sets the status for rx & tx (bits STAT_TX[1:0] & STAT_RX[1:0])

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wStaterx: new state.
- wStatetx: new state.

Return value:

- None

PCD_GET_EP_TX_STATUS**Description:**

- gets the status for tx/rx transfer (bits STAT_TX[1:0] /STAT_RX[1:0])

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- status

PCD_GET_EP_RX_STATUS**PCD_SET_EP_TX_VALID****Description:**

- sets directly the VALID tx/rx-status into the endpoint register

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_SET_EP_RX_VALID**PCD_GET_EP_TX_STALL_STATUS****Description:**

- checks stall condition in an endpoint.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- TRUE: = endpoint in stall condition.

PCD_GET_EP_RX_STALL_STATUS**PCD_SET_EP_KIND****Description:**

- set & clear EP_KIND bit.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_EP_KIND**PCD_SET_OUT_STATUS****Description:**

- Sets/clears directly STATUS_OUT bit in the endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_OUT_STATUS**PCD_SET_EP_DBUF****Description:**

- Sets/clears directly EP_KIND bit in the endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_EP_DBUF**PCD_CLEAR_RX_EP_CTR****Description:**

- Clears bit CTR_RX / CTR_TX in the endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_TX_EP_CTR

PCD_RX_DTOG

Description:

- Toggles DTOG_RX / DTOG_TX bit in the endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_TX_DTOG

PCD_CLEAR_RX_DTOG

Description:

- Clears DTOG_RX / DTOG_TX bit in the endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_CLEAR_TX_DTOG

PCD_SET_EP_ADDRESS

Description:

- Sets address in an endpoint register.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- bAddr: Address.

Return value:

- None

PCD_GET_EP_ADDRESS

PCD_EP_TX_ADDRESS

PCD_EP_TX_CNT

PCD_EP_RX_ADDRESS

PCD_EP_RX_CNT

PCD_SET_EP_RX_CNT

PCD_SET_EP_TX_ADDRESS

Description:

- sets address of the tx/rx buffer.

Parameters:

PCD_SET_EP_RX_ADDRESS

PCD_GET_EP_TX_ADDRESS

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wAddr: address to be set (must be word aligned).

Return value:

- None

Description:

- Gets address of the tx/rx buffer.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- address: of the buffer.

PCD_GET_EP_RX_ADDRESS

PCD_CALC_BLK32

Description:

- Sets counter of rx buffer with no.

Parameters:

- dwReg: Register
- wCount: Counter.
- wNBlocks: no. of Blocks.

Return value:

- None

PCD_CALC_BLK2

PCD_SET_EP_CNT_RX_REG

PCD_SET_EP_RX_DBUF0_CNT

PCD_SET_EP_TX_CNT

Description:

- sets counter for the tx/rx buffer.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wCount: Counter value.

Return value:

- None

PCD_GET_EP_TX_CNT

Description:

- gets counter of the tx buffer.

PCD_GET_EP_RX_CNT

PCD_SET_EP_DBUF0_ADDR

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- Counter: value

Description:

- Sets buffer 0/1 address in a double buffer endpoint.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.

Return value:

- Counter: value

PCD_SET_EP_DBUF1_ADDR

PCD_SET_EP_DBUF_ADDR

Description:

- Sets addresses in a double buffer endpoint.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- wBuf0Addr: buffer 0 address.
- wBuf1Addr: = buffer 1 address.

Return value:

- None

PCD_GET_EP_DBUF0_ADDR

Description:

- Gets buffer 0/1 address of a double buffer endpoint.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_GET_EP_DBUF1_ADDR

PCD_SET_EP_DBUF0_CNT

Description:

- Gets buffer 0/1 address of a double buffer

endpoint.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.
- bDir: endpoint dir EP_DBUF_OUT = OUT
EP_DBUF_IN = IN
- wCount: Counter value

Return value:

- None

PCD_SET_EP_DBUF1_CNT

PCD_SET_EP_DBUF_CNT

PCD_GET_EP_DBUF0_CNT

Description:

- Gets buffer 0/1 rx/tx counter for double buffering.

Parameters:

- USBx: USB peripheral instance register address.
- bEpNum: Endpoint Number.

Return value:

- None

PCD_GET_EP_DBUF1_CNT

30 HAL PCD Extension Driver

30.1 HAL PCD Extension Driver

30.2 PCDEx Firmware driver API description

30.2.1 Peripheral Control functions

This section provides functions allowing to:

- Configure PMA for the EndPoint

This section contains the following APIs:

- [*HAL_PCDEx_PMAConfig\(\)*](#)

30.2.2 HAL_PCDEx_PMAConfig

Function Name `HAL_StatusTypeDef HAL_PCDEx_PMAConfig
(PCD_HandleTypeDef * hpcd, uint16_t ep_addr, uint16_t
ep_kind, uint32_t pmaaddress)`

Function Description Configure PMA for EP.

Parameters

- **hpcd:** : Device instance
- **ep_addr:** endpoint address
- **ep_kind:** endpoint Kind USB_SNG_BUF: Single Buffer used
USB_DBL_BUF: Double Buffer used
- **pmaaddress:** EP address in The PMA: In case of single buffer
endpoint this parameter is 16-bit value providing the address
in PMA allocated to endpoint. In case of double buffer
endpoint this parameter is a 32-bit value providing the
endpoint buffer 0 address in the LSB part of 32-bit value and
endpoint buffer 1 address in the MSB part of 32-bit value.

Return values

- : status

30.3 PCDEx Firmware driver defines

30.3.1 PCDEx

31 HAL PWR Generic Driver

31.1 HAL PWR Generic Driver

31.2 PWR Firmware driver registers structures

31.2.1 PWR_PVDTypeDef

Data Fields

- *uint32_t PVDLevel*
- *uint32_t Mode*

Field Documentation

- *uint32_t PWR_PVDTypeDef::PVDLevel*
PVDLevel: Specifies the PVD detection level. This parameter can be a value of [PWR_PVD_detection_level](#)
- *uint32_t PWR_PVDTypeDef::Mode*
Mode: Specifies the operating mode for the selected pins. This parameter can be a value of [PWR_PVD_Mode](#)

31.3 PWR Firmware driver API description

31.3.1 Initialization and de-initialization functions

After reset, the backup domain (RTC registers, RTC backup data registers) is protected against possible unwanted write accesses. To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the `__HAL_RCC_PWR_CLK_ENABLE()` macro.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.

This section contains the following APIs:

- [HAL_PWR_DeInit\(\)](#)
- [HAL_PWR_EnableBkUpAccess\(\)](#)
- [HAL_PWR_DisableBkUpAccess\(\)](#)

31.3.2 Peripheral Control functions

PVD configuration

- The PVD is used to monitor the VDD power supply by comparing it to a threshold selected by the PVD Level (PLS[2:0] bits in the `PWR_CR`).
- The PVD can use an external input analog voltage (`PVD_IN`) which is compared internally to `VREFINT`. The `PVD_IN` (PB7) has to be configured in Analog mode when `PWR_PVDLevel_7` is selected (PLS[2:0] = 111).

- A PVDO flag is available to indicate if VDD/VDDA is higher or lower than the PVD threshold. This event is internally connected to the EXTI line16 and can generate an interrupt if enabled. This is done through `__HAL_PWR_PVD_EXTI_ENABLE_IT()` macro.
- The PVD is stopped in Standby mode.

WakeUp pin configuration

- WakeUp pin is used to wake up the system from Standby mode. This pin is forced in input pull-down configuration and is active on rising edges.
- There are two or three WakeUp pins: WakeUp Pin 1 on PA.00. WakeUp Pin 2 on PC.13. WakeUp Pin 3 on PE.06. : Only on product with GPIOE available

Main and Backup Regulators configuration

Low Power modes configuration

The device features 5 low-power modes:

- Low power run mode: regulator in low power mode, limited clock frequency, limited number of peripherals running.
- Sleep mode: Cortex-M3 core stopped, peripherals kept running.
- Low power sleep mode: Cortex-M3 core stopped, limited clock frequency, limited number of peripherals running, regulator in low power mode.
- Stop mode: All clocks are stopped, regulator running, regulator in low power mode.
- Standby mode: VCore domain powered off

Low power run mode

To further reduce the consumption when the system is in Run mode, the regulator can be configured in low power mode. In this mode, the system frequency should not exceed MSI frequency range1. In Low power run mode, all I/O pins keep the same state as in Run mode.

- Entry:
 - VCore in range2
 - Decrease the system frequency to not exceed the frequency of MSI frequency range1.
 - The regulator is forced in low power mode using the `HAL_PWREx_EnableLowPowerRunMode()` function.
- Exit:
 - The regulator is forced in Main regulator mode using the `HAL_PWREx_DisableLowPowerRunMode()` function.
 - Increase the system frequency if needed.

Sleep mode

- Entry: The Sleep mode is entered by using the `HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFx)` functions with
 - `PWR_SLEEPENTRY_WFI`: enter SLEEP mode with WFI instruction
 - `PWR_SLEEPENTRY_WFE`: enter SLEEP mode with WFE instruction

- Exit:
 - Any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Sleep mode.

Low power sleep mode

- Entry: The Low power sleep mode is entered by using the `HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFx)` functions with
 - `PWR_SLEEPENTRY_WFI`: enter SLEEP mode with WFI instruction
 - `PWR_SLEEPENTRY_WFE`: enter SLEEP mode with WFE instruction
- The Flash memory can be switched off by using the control bits (`SLEEP_PD` in the `FLASH_ACR` register). This reduces power consumption but increases the wake-up time.
- Exit:
 - If the WFI instruction was used to enter Low power sleep mode, any peripheral interrupt acknowledged by the nested vectored interrupt controller (NVIC) can wake up the device from Low power sleep mode. If the WFE instruction was used to enter Low power sleep mode, the MCU exits Sleep mode as soon as an event occurs.

Stop mode

The Stop mode is based on the Cortex-M3 deepsleep mode combined with peripheral clock gating. The voltage regulator can be configured either in normal or low-power mode. In Stop mode, all clocks in the VCore domain are stopped, the PLL, the MSI, the HSI and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. To get the lowest consumption in Stop mode, the internal Flash memory also enters low power mode. When the Flash memory is in power-down mode, an additional startup delay is incurred when waking up from Stop mode. To minimize the consumption In Stop mode, VREFINT, the BOR, PVD, and temperature sensor can be switched off before entering Stop mode. They can be switched on again by software after exiting Stop mode using the ULP bit in the `PWR_CR` register. In Stop mode, all I/O pins keep the same state as in Run mode.

- Entry: The Stop mode is entered using the `HAL_PWR_EnterSTOPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI)` function with:
 - Main regulator ON.
 - Low Power regulator ON.
 - `PWR_SLEEPENTRY_WFI`: enter SLEEP mode with WFI instruction
 - `PWR_SLEEPENTRY_WFE`: enter SLEEP mode with WFE instruction
- Exit:
 - By issuing an interrupt or a wakeup event, the MSI RC oscillator is selected as system clock.

Standby mode

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex-M3 deepsleep mode, with the voltage regulator disabled. The VCore domain is consequently powered off. The PLL, the MSI, the HSI oscillator and the HSE oscillator are also switched off. SRAM and register contents are lost except for the RTC registers, RTC backup registers and Standby circuitry. To minimize the consumption In Standby mode, VREFINT, the BOR, PVD, and temperature sensor can be switched off before entering the

Standby mode. They can be switched on again by software after exiting the Standby mode. function.

- Entry:
 - The Standby mode is entered using the HAL_PWR_EnterSTANDBYMode() function.
- Exit:
 - WKUP pin rising edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time-stamp event, external reset in NRST pin, IWDG reset.

Auto-wakeup (AWU) from low-power mode

The MCU can be woken up from low-power mode by an RTC Alarm event, an RTC Wakeup event, a tamper event, a time-stamp event, or a comparator event, without depending on an external interrupt (Auto-wakeup mode).

- RTC auto-wakeup (AWU) from the Stop mode
 - To wake up from the Stop mode with an RTC alarm event, it is necessary to:
 - Configure the EXTI Line 17 to be sensitive to rising edges (Interrupt or Event modes) and Enable the RTC Alarm Interrupt using the HAL_RTC_SetAlarm_IT() function
 - Configure the RTC to generate the RTC alarm using the HAL_RTC_Init() and HAL_RTC_SetTime() functions.
 - To wake up from the Stop mode with an RTC Tamper or time stamp event, it is necessary to:
 - Configure the EXTI Line 19 to be sensitive to rising edges (Interrupt or Event modes) and Enable the RTC Tamper or time stamp Interrupt using the HAL_RTCEX_SetTamper_IT() or HAL_RTCEX_SetTimeStamp_IT() functions.
 - To wake up from the Stop mode with an RTC WakeUp event, it is necessary to:
 - Configure the EXTI Line 20 to be sensitive to rising edges (Interrupt or Event modes) and Enable the RTC WakeUp Interrupt using the HAL_RTCEX_SetWakeUpTimer_IT() function.
 - Configure the RTC to generate the RTC WakeUp event using the HAL_RTCEX_SetWakeUpTimer() function.
- RTC auto-wakeup (AWU) from the Standby mode
 - To wake up from the Standby mode with an RTC alarm event, it is necessary to:
 - Enable the RTC Alarm Interrupt using the HAL_RTC_SetAlarm_IT() function.
 - Configure the RTC to generate the RTC alarm using the HAL_RTC_Init() and HAL_RTC_SetTime() functions.
 - To wake up from the Standby mode with an RTC Tamper or time stamp event, it is necessary to:
 - Enable the RTC Tamper or time stamp Interrupt and Configure the RTC to detect the tamper or time stamp event using the HAL_RTCEX_SetTimeStamp_IT() or HAL_RTCEX_SetTamper_IT() functions.
 - To wake up from the Standby mode with an RTC WakeUp event, it is necessary to:
 - Enable the RTC WakeUp Interrupt and Configure the RTC to generate the RTC WakeUp event using the HAL_RTCEX_SetWakeUpTimer_IT() and HAL_RTCEX_SetWakeUpTimer() functions.
- Comparator auto-wakeup (AWU) from the Stop mode
 - To wake up from the Stop mode with an comparator 1 or comparator 2 wakeup event, it is necessary to:

- Configure the EXTI Line 21 or EXTI Line 22 for comparator to be sensitive to the selected edges (falling, rising or falling and rising) (Interrupt or Event modes) using the COMP functions.
- Configure the comparator to generate the event.

This section contains the following APIs:

- [*HAL_PWR_ConfigPVD\(\)*](#)
- [*HAL_PWR_EnablePVD\(\)*](#)
- [*HAL_PWR_DisablePVD\(\)*](#)
- [*HAL_PWR_EnableWakeUpPin\(\)*](#)
- [*HAL_PWR_DisableWakeUpPin\(\)*](#)
- [*HAL_PWR_EnterSLEEPMode\(\)*](#)
- [*HAL_PWR_EnterSTOPMode\(\)*](#)
- [*HAL_PWR_EnterSTANDBYMode\(\)*](#)
- [*HAL_PWR_EnableSleepOnExit\(\)*](#)
- [*HAL_PWR_DisableSleepOnExit\(\)*](#)
- [*HAL_PWR_EnableSEVOnPend\(\)*](#)
- [*HAL_PWR_DisableSEVOnPend\(\)*](#)
- [*HAL_PWR_PVD_IRQHandler\(\)*](#)
- [*HAL_PWR_PVDCallback\(\)*](#)

31.3.3 HAL_PWR_DeInit

Function Name	void HAL_PWR_DeInit (void)
Function Description	Deinitializes the PWR peripheral registers to their default reset values.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • Before calling this function, the VOS[1:0] bits should be configured to "10" and the system frequency has to be configured accordingly. To configure the VOS[1:0] bits, use the PWR_VoltageScalingConfig() function. • ULP and FWU bits are not reset by this function.

31.3.4 HAL_PWR_EnableBkUpAccess

Function Name	void HAL_PWR_EnableBkUpAccess (void)
Function Description	Enables access to the backup domain (RTC registers, RTC backup data registers).
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock, the Backup Domain Access should be kept enabled.

31.3.5 HAL_PWR_DisableBkUpAccess

Function Name	void HAL_PWR_DisableBkUpAccess (void)
Function Description	Disables access to the backup domain (RTC registers, RTC backup data registers).
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • If the HSE divided by 2, 4, 8 or 16 is used as the RTC clock,

the Backup Domain Access should be kept enabled.

31.3.6 HAL_PWR_ConfigPVD

Function Name	void HAL_PWR_ConfigPVD (PWR_PVDTypeDef * sConfigPVD)
Function Description	Configures the voltage threshold detected by the Power Voltage Detector(PVD).
Parameters	<ul style="list-style-type: none"> • sConfigPVD: pointer to an PWR_PVDTypeDef structure that contains the configuration information for the PVD.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • Refer to the electrical characteristics of your device datasheet for more details about the voltage threshold corresponding to each detection level.

31.3.7 HAL_PWR_EnablePVD

Function Name	void HAL_PWR_EnablePVD (void)
Function Description	Enables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> • None

31.3.8 HAL_PWR_DisablePVD

Function Name	void HAL_PWR_DisablePVD (void)
Function Description	Disables the Power Voltage Detector(PVD).
Return values	<ul style="list-style-type: none"> • None

31.3.9 HAL_PWR_EnableWakeUpPin

Function Name	void HAL_PWR_EnableWakeUpPin (uint32_t WakeUpPinx)
Function Description	Enables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> • WakeUpPinx: Specifies the Power Wake-Up pin to enable. This parameter can be one of the following values: PWR_WAKEUP_PIN1PWR_WAKEUP_PIN2PWR_WAKEUP_PIN3: Only on product with GPIOE available
Return values	<ul style="list-style-type: none"> • None

31.3.10 HAL_PWR_DisableWakeUpPin

Function Name	void HAL_PWR_DisableWakeUpPin (uint32_t WakeUpPinx)
Function Description	Disables the WakeUp PINx functionality.
Parameters	<ul style="list-style-type: none"> • WakeUpPinx: Specifies the Power Wake-Up pin to disable. This parameter can be one of the following values: PWR_WAKEUP_PIN1PWR_WAKEUP_PIN2PWR_WAKEUP_PIN3: Only on product with GPIOE available
Return values	<ul style="list-style-type: none"> • None

31.3.11 HAL_PWR_EnterSLEEPMode

Function Name	void HAL_PWR_EnterSLEEPMode (uint32_t Regulator, uint8_t SLEEPEntry)
Function Description	Enters Sleep mode.
Parameters	<ul style="list-style-type: none"> • Regulator: Specifies the regulator state in SLEEP mode. This parameter can be one of the following values: PWR_MAINREGULATOR_ON: SLEEP mode with regulator ON ONPWR_LOWPOWERREGULATOR_ON: SLEEP mode with low power regulator ON • SLEEPEntry: Specifies if SLEEP mode is entered with WFI or WFE instruction. When WFI entry is used, tick interrupt have to be disabled if not desired as the interrupt wake up source. This parameter can be one of the following values: PWR_SLEEPENTRY_WFI: enter SLEEP mode with WFI instruction PWR_SLEEPENTRY_WFE: enter SLEEP mode with WFE instruction
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • In Sleep mode, all I/O pins keep the same state as in Run mode.

31.3.12 HAL_PWR_EnterSTOPMode

Function Name	void HAL_PWR_EnterSTOPMode (uint32_t Regulator, uint8_t STOPEntry)
Function Description	Enters Stop mode.
Parameters	<ul style="list-style-type: none"> • Regulator: Specifies the regulator state in Stop mode. This parameter can be one of the following values: PWR_MAINREGULATOR_ON: Stop mode with regulator ON ONPWR_LOWPOWERREGULATOR_ON: Stop mode with low power regulator ON • STOPEntry: Specifies if Stop mode is entered with WFI or WFE instruction. This parameter can be one of the following values: PWR_STOPENTRY_WFI: Enter Stop mode with WFI instruction PWR_STOPENTRY_WFE: Enter Stop mode with WFE instruction
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • In Stop mode, all I/O pins keep the same state as in Run mode. • When exiting Stop mode by using an interrupt or a wakeup event, MSI RC oscillator is selected as system clock. • When the voltage regulator operates in low power mode, an additional startup delay is incurred when waking up from Stop mode. By keeping the internal regulator ON during Stop mode, the consumption is higher although the startup time is reduced.

31.3.13 HAL_PWR_EnterSTANDBYMode

Function Name	void HAL_PWR_EnterSTANDBYMode (void)
---------------	--

Function Description	Enters Standby mode.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> In Standby mode, all I/O pins are high impedance except for: Reset pad (still available) RTC_AF1 pin (PC13) if configured for tamper, time-stamp, RTC Alarm out, or RTC clock calibration out. WKUP pin 1 (PA0) if enabled. WKUP pin 2 (PC13) if enabled. WKUP pin 3 (PE6) if enabled.

31.3.14 HAL_PWR_EnableSleepOnExit

Function Name	void HAL_PWR_EnableSleepOnExit (void)
Function Description	Indicates Sleep-On-Exit when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over. Setting this bit is useful when the processor is expected to run only on interruptions handling.

31.3.15 HAL_PWR_DisableSleepOnExit

Function Name	void HAL_PWR_DisableSleepOnExit (void)
Function Description	Disables Sleep-On-Exit feature when returning from Handler mode to Thread mode.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> Clears SLEEPONEXIT bit of SCR register. When this bit is set, the processor re-enters SLEEP mode when an interruption handling is over.

31.3.16 HAL_PWR_EnableSEVOnPend

Function Name	void HAL_PWR_EnableSEVOnPend (void)
Function Description	Enables CORTEX M3 SEVONPEND bit.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> Sets SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

31.3.17 HAL_PWR_DisableSEVOnPend

Function Name	void HAL_PWR_DisableSEVOnPend (void)
Function Description	Disables CORTEX M3 SEVONPEND bit.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> Clears SEVONPEND bit of SCR register. When this bit is set, this causes WFE to wake up when an interrupt moves from inactive to pended.

31.3.18 HAL_PWR_PVD_IRQHandler

Function Name	void HAL_PWR_PVD_IRQHandler (void)
Function Description	This function handles the PWR PVD interrupt request.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> This API should be called under the PVD_IRQHandler().

31.3.19 HAL_PWR_PVDCallback

Function Name	void HAL_PWR_PVDCallback (void)
Function Description	PWR PVD interrupt callback.
Return values	<ul style="list-style-type: none"> None

31.4 PWR Firmware driver defines**31.4.1 PWR*****PWR CR Register alias address***

LPSSDR_BIT_NUMBER

CR_LPSSDR_BB

DBP_BIT_NUMBER

CR_DBP_BB

LPRUN_BIT_NUMBER

CR_LPRUN_BB

PVDE_BIT_NUMBER

CR_PVDE_BB

FWU_BIT_NUMBER

CR_FWU_BB

ULP_BIT_NUMBER

CR_ULP_BB

PWR CSR Register alias address

CSR_EWUP_BB

PWR Exported Macros**__HAL_PWR_VOLTAGESCALING_CONFIG****Description:**

- macros configure the main internal regulator output voltage.

Parameters:

- __REGULATOR__**: specifies the regulator output voltage to achieve a tradeoff between performance and power consumption when the device does not operate at the maximum

frequency (refer to the datasheets for more details). This parameter can be one of the following values:

- PWR_REGULATOR_VOLTAGE_SCALE1: Regulator voltage output Scale 1 mode, System frequency up to 32 MHz.
- PWR_REGULATOR_VOLTAGE_SCALE2: Regulator voltage output Scale 2 mode, System frequency up to 16 MHz.
- PWR_REGULATOR_VOLTAGE_SCALE3: Regulator voltage output Scale 3 mode, System frequency up to 4.2 MHz

Return value:

- None

Description:

- Check PWR flag is set or not.

Parameters:

- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - PWR_FLAG_WU: Wake Up flag. This flag indicates that a wakeup event was received from the WKUP pin or from the RTC alarm (Alarm B), RTC Tamper event, RTC TimeStamp event or RTC Wakeup. An additional wakeup event is detected if the WKUP pin is enabled (by setting the EWUP bit) when the WKUP pin level is already high.
 - PWR_FLAG_SB: StandBy flag. This flag indicates that the system was resumed from StandBy mode.
 - PWR_FLAG_PVDO: PVD Output. This flag is valid only if PVD is enabled by the `HAL_PWR_EnablePVD()` function. The PVD is stopped by Standby mode. For this reason, this bit is equal to 0 after Standby or reset until the PVDE bit is set.
 - PWR_FLAG_VREFINTRDY: Internal voltage reference (VREFINT) ready flag. This bit indicates the state of the

`__HAL_PWR_GET_FLAG`

internal voltage reference, VREFINT.

- PWR_FLAG_VOS: Voltage Scaling select flag. A delay is required for the internal regulator to be ready after the voltage range is changed. The VOSF bit indicates that the regulator has reached the voltage level defined with bits VOS of PWR_CR register.
- PWR_FLAG_REGLP: Regulator LP flag. When the MCU exits from Low power run mode, this bit stays at 1 until the regulator is ready in main mode. A polling on this bit is recommended to wait for the regulator main mode. This bit is reset by hardware when the regulator is ready.

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

Description:

- Clear the PWR's pending flags.

Parameters:

- __FLAG__: specifies the flag to clear. This parameter can be one of the following values:
 - PWR_FLAG_WU: Wake Up flag
 - PWR_FLAG_SB: StandBy flag

Description:

- Enable interrupt on PVD Exti Line 16.

Return value:

- None.

Description:

- Disable interrupt on PVD Exti Line 16.

Return value:

- None.

Description:

- Enable event on PVD Exti Line 16.

Return value:

`__HAL_PWR_CLEAR_FLAG`

`__HAL_PWR_PVD_EXTI_ENABLE_IT`

`__HAL_PWR_PVD_EXTI_DISABLE_IT`

`__HAL_PWR_PVD_EXTI_ENABLE_EVENT`

__HAL_PWR_PVD_EXTI_DISABLE_EVENT	<ul style="list-style-type: none">• None. Description: <ul style="list-style-type: none">• Disable event on PVD Exti Line 16. Return value: <ul style="list-style-type: none">• None.
__HAL_PWR_PVD_EXTI_ENABLE_FALLING_EDGE	Description: <ul style="list-style-type: none">• PVD EXTI line configuration: set falling edge trigger. Return value: <ul style="list-style-type: none">• None.
__HAL_PWR_PVD_EXTI_DISABLE_FALLING_EDGE	Description: <ul style="list-style-type: none">• Disable the PVD Extended Interrupt Falling Trigger. Return value: <ul style="list-style-type: none">• None.
__HAL_PWR_PVD_EXTI_ENABLE_RISING_EDGE	Description: <ul style="list-style-type: none">• PVD EXTI line configuration: set rising edge trigger. Return value: <ul style="list-style-type: none">• None.
__HAL_PWR_PVD_EXTI_DISABLE_RISING_EDGE	Description: <ul style="list-style-type: none">• Disable the PVD Extended Interrupt Rising Trigger. Return value: <ul style="list-style-type: none">• None.
__HAL_PWR_PVD_EXTI_ENABLE_RISING_FALLING_EDGE	Description: <ul style="list-style-type: none">• PVD EXTI line configuration: set rising & falling edge trigger. Return value: <ul style="list-style-type: none">• None.
__HAL_PWR_PVD_EXTI_DISABLE_RISING_FALLING_EDGE	Description: <ul style="list-style-type: none">• Disable the PVD Extended Interrupt Rising & Falling Trigger. Return value: <ul style="list-style-type: none">• None.
__HAL_PWR_PVD_EXTI_GET_FLAG	Description: <ul style="list-style-type: none">• Check whether the specified PVD EXTI interrupt flag is set or not.

__HAL_PWR_PVD_EXTI_CLEAR_FLAG

Return value:

- EXTI: PVD Line Status.

Description:

- Clear the PVD EXTI flag.

Return value:

- None.

__HAL_PWR_PVD_EXTI_GENERATE_SWIT

Description:

- Generate a Software interrupt on selected EXTI line.

Return value:

- None.

PWR Flag

PWR_FLAG_WU

PWR_FLAG_SB

PWR_FLAG_PVDO

PWR_FLAG_VREFINTRDY

PWR_FLAG_VOS

PWR_FLAG_REGLP

PWR Private Constants

PWR_EXTI_LINE_PVD External interrupt line 16 Connected to the PVD EXTI Line

PWR Private Macros

IS_PWR_PVD_LEVEL

IS_PWR_PVD_MODE

IS_PWR_REGULATOR

IS_PWR_SLEEP_ENTRY

IS_PWR_STOP_ENTRY

IS_PWR_VOLTAGE_SCALING_RANGE

PWR PVD detection level

PWR_PVDLEVEL_0

PWR_PVDLEVEL_1

PWR_PVDLEVEL_2

PWR_PVDLEVEL_3

PWR_PVDLEVEL_4

PWR_PVDLEVEL_5

PWR_PVDLEVEL_6

PWR_PVDLEVEL_7

PWR PVD Mode

PWR_PVD_MODE_NORMAL	basic mode is used
PWR_PVD_MODE_IT_RISING	External Interrupt Mode with Rising edge trigger detection
PWR_PVD_MODE_IT_FALLING	External Interrupt Mode with Falling edge trigger detection
PWR_PVD_MODE_IT_RISING_FALLING	External Interrupt Mode with Rising/Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING	Event Mode with Rising edge trigger detection
PWR_PVD_MODE_EVENT_FALLING	Event Mode with Falling edge trigger detection
PWR_PVD_MODE_EVENT_RISING_FALLING	Event Mode with Rising/Falling edge trigger detection

PWR Register alias address

PWR_OFFSET

PWR_CR_OFFSET

PWR_CSR_OFFSET

PWR_CR_OFFSET_BB

PWR_CSR_OFFSET_BB

PWR Regulator state in SLEEP/STOP mode

PWR_MAINREGULATOR_ON

PWR_LOWPOWERREGULATOR_ON

PWR Regulator Voltage Scale

PWR_REGULATOR_VOLTAGE_SCALE1

PWR_REGULATOR_VOLTAGE_SCALE2

PWR_REGULATOR_VOLTAGE_SCALE3

PWR SLEEP mode entry

PWR_SLEEPENTRY_WFI

PWR_SLEEPENTRY_WFE

PWR STOP mode entry

PWR_STOPENTRY_WFI

PWR_STOPENTRY_WFE

32 HAL PWR Extension Driver

32.1 HAL PWR Extension Driver

32.2 PWREx Firmware driver API description

32.2.1 Peripheral extended features functions

This section contains the following APIs:

- [HAL_PWREx_GetVoltageRange\(\)](#)
- [HAL_PWREx_EnableFastWakeUp\(\)](#)
- [HAL_PWREx_DisableFastWakeUp\(\)](#)
- [HAL_PWREx_EnableUltraLowPower\(\)](#)
- [HAL_PWREx_DisableUltraLowPower\(\)](#)
- [HAL_PWREx_EnableLowPowerRunMode\(\)](#)
- [HAL_PWREx_DisableLowPowerRunMode\(\)](#)

32.2.2 HAL_PWREx_GetVoltageRange

Function Name	uint32_t HAL_PWREx_GetVoltageRange (void)
Function Description	Return Voltage Scaling Range.
Return values	<ul style="list-style-type: none">• VOS bit field (PWR_REGULATOR_VOLTAGE_SCALE1, PWR_REGULATOR_VOLTAGE_SCALE2 or PWR_REGULATOR_VOLTAGE_SCALE3)

32.2.3 HAL_PWREx_EnableFastWakeUp

Function Name	void HAL_PWREx_EnableFastWakeUp (void)
Function Description	Enables the Fast WakeUp from Ultra Low Power mode.
Return values	<ul style="list-style-type: none">• None
Notes	<ul style="list-style-type: none">• This bit works in conjunction with ULP bit. Means, when ULP = 1 and FWU = 1 :VREFINT startup time is ignored when exiting from low power mode.

32.2.4 HAL_PWREx_DisableFastWakeUp

Function Name	void HAL_PWREx_DisableFastWakeUp (void)
Function Description	Disables the Fast WakeUp from Ultra Low Power mode.
Return values	<ul style="list-style-type: none">• None

32.2.5 HAL_PWREx_EnableUltraLowPower

Function Name	void HAL_PWREx_EnableUltraLowPower (void)
Function Description	Enables the Ultra Low Power mode.
Return values	<ul style="list-style-type: none">• None

32.2.6 HAL_PWREx_DisableUltraLowPower

Function Name	void HAL_PWREx_DisableUltraLowPower (void)
Function Description	Disables the Ultra Low Power mode.
Return values	<ul style="list-style-type: none">• None

32.2.7 HAL_PWREx_EnableLowPowerRunMode

Function Name	void HAL_PWREx_EnableLowPowerRunMode (void)
Function Description	Enters the Low Power Run mode.
Return values	<ul style="list-style-type: none">• None
Notes	<ul style="list-style-type: none">• Low power run mode can only be entered when VCORE is in range 2. In addition, the dynamic voltage scaling must not be used when Low power run mode is selected. Only Stop and Sleep modes with regulator configured in Low power mode is allowed when Low power run mode is selected.• In Low power run mode, all I/O pins keep the same state as in Run mode.

32.2.8 HAL_PWREx_DisableLowPowerRunMode

Function Name	void HAL_PWREx_DisableLowPowerRunMode (void)
Function Description	Exits the Low Power Run mode.
Return values	<ul style="list-style-type: none">• None

32.3 PWREx Firmware driver defines

32.3.1 PWREx

PWREx Wakeup Pins

PWR_WAKEUP_PIN1

PWR_WAKEUP_PIN2

PWR_WAKEUP_PIN3

IS_PWR_WAKEUP_PIN

33 HAL RCC Generic Driver

33.1 HAL RCC Generic Driver

33.2 RCC Firmware driver registers structures

33.2.1 RCC_PLLInitTypeDef

Data Fields

- *uint32_t PLLState*
- *uint32_t PLLSource*
- *uint32_t PLLMUL*
- *uint32_t PLLDIV*

Field Documentation

- *uint32_t RCC_PLLInitTypeDef::PLLState*
The new state of the PLL. This parameter can be a value of [RCC_PLL_Config](#)
- *uint32_t RCC_PLLInitTypeDef::PLLSource*
PLLSource: PLL entry clock source. This parameter must be a value of [RCC_PLL_Clock_Source](#)
- *uint32_t RCC_PLLInitTypeDef::PLLMUL*
PLLMUL: Multiplication factor for PLL VCO input clock This parameter must be a value of [RCC_PLL_Multiplication_Factor](#)
- *uint32_t RCC_PLLInitTypeDef::PLLDIV*
PLLDIV: Division factor for PLL VCO input clock This parameter must be a value of [RCC_PLL_Division_Factor](#)

33.2.2 RCC_OscInitTypeDef

Data Fields

- *uint32_t OscillatorType*
- *uint32_t HSEState*
- *uint32_t LSEState*
- *uint32_t HSISState*
- *uint32_t HSICalibrationValue*
- *uint32_t LSISState*
- *uint32_t MSISState*
- *uint32_t MSICalibrationValue*
- *uint32_t MSIClockRange*
- *RCC_PLLInitTypeDef PLL*

Field Documentation

- ***uint32_t RCC_OscInitTypeDef::OscillatorType***
The oscillators to be configured. This parameter can be a value of [RCC_Oscillator_Type](#)
- ***uint32_t RCC_OscInitTypeDef::HSEState***
The new state of the HSE. This parameter can be a value of [RCC_HSE_Config](#)
- ***uint32_t RCC_OscInitTypeDef::LSEState***
The new state of the LSE. This parameter can be a value of [RCC_LSE_Config](#)
- ***uint32_t RCC_OscInitTypeDef::HSIState***
The new state of the HSI. This parameter can be a value of [RCC_HSI_Config](#)
- ***uint32_t RCC_OscInitTypeDef::HSICalibrationValue***
The HSI calibration trimming value (default is RCC_HSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x1F
- ***uint32_t RCC_OscInitTypeDef::LSIState***
The new state of the LSI. This parameter can be a value of [RCC_LSI_Config](#)
- ***uint32_t RCC_OscInitTypeDef::MSIState***
The new state of the MSI. This parameter can be a value of [RCC_MSI_Config](#)
- ***uint32_t RCC_OscInitTypeDef::MSICalibrationValue***
The MSI calibration trimming value. (default is RCC_MSICALIBRATION_DEFAULT). This parameter must be a number between Min_Data = 0x00 and Max_Data = 0xFF
- ***uint32_t RCC_OscInitTypeDef::MSIClockRange***
The MSI frequency range. This parameter can be a value of [RCC_MSI_Clock_Range](#)
- ***RCC_PLLInitTypeDef RCC_OscInitTypeDef::PLL***
PLL structure parameters

33.2.3 RCC_ClkInitTypeDef

Data Fields

- ***uint32_t ClockType***
- ***uint32_t SYSCLKSource***
- ***uint32_t AHBCLKDivider***
- ***uint32_t APB1CLKDivider***
- ***uint32_t APB2CLKDivider***

Field Documentation

- ***uint32_t RCC_ClkInitTypeDef::ClockType***
The clock to be configured. This parameter can be a value of [RCC_System_Clock_Type](#)
- ***uint32_t RCC_ClkInitTypeDef::SYSCLKSource***
The clock source (SYSCLKS) used as system clock. This parameter can be a value of [RCC_System_Clock_Source](#)
- ***uint32_t RCC_ClkInitTypeDef::AHBCLKDivider***
The AHB clock (HCLK) divider. This clock is derived from the system clock (SYSCLK). This parameter can be a value of [RCC_AHB_Clock_Source](#)
- ***uint32_t RCC_ClkInitTypeDef::APB1CLKDivider***
The APB1 clock (PCLK1) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of [RCC_APB1_APB2_Clock_Source](#)
- ***uint32_t RCC_ClkInitTypeDef::APB2CLKDivider***
The APB2 clock (PCLK2) divider. This clock is derived from the AHB clock (HCLK). This parameter can be a value of [RCC_APB1_APB2_Clock_Source](#)

33.3 RCC Firmware driver API description

33.3.1 RCC specific features

After reset the device is running from multispeed internal oscillator clock (MSI 2.097MHz) with Flash 0 wait state and Flash prefetch buffer is disabled, and all peripherals are off except internal SRAM, Flash and JTAG.

- There is no prescaler on High speed (AHB) and Low speed (APB) busses; all peripherals mapped on these busses are running at MSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.
- All GPIOs are in input floating state, except the JTAG pins which are assigned to be used for debug purpose.

Once the device started from reset, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB busses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals whose clocks are not derived from the System clock (I2S, RTC, ADC, USB OTG FS/SDIO/RNG) (*) SDIO only for STM32L1xxxD devices

33.3.2 RCC Limitations

A delay between an RCC peripheral clock enable and the effective peripheral enabling should be taken into account in order to manage the peripheral read/write from/to registers.

- This delay depends on the peripheral mapping.
 - AHB & APB peripherals, 1 dummy read is necessary

Workarounds:

1. For AHB & APB peripherals, a dummy read to the peripheral register has been inserted in each `__HAL_RCC_PPP_CLK_ENABLE()` macro.

33.3.3 Initialization and de-initialization function

This section provides functions allowing to configure the internal/external oscillators (MSI, HSE, HSI, LSE, LSI, PLL, CSS and MCO) and the System busses clocks (SYSCCLK, AHB, APB1 and APB2).

Internal/external clock and PLL configuration

1. MSI (Multispeed internal), Seven frequency ranges are available: 65.536 kHz, 131.072 kHz, 262.144 kHz, 524.288 kHz, 1.048 MHz, 2.097 MHz (default value) and 4.194 MHz.
2. HSI (high-speed internal), 16 MHz factory-trimmed RC used directly or through the PLL as System clock source.
3. LSI (low-speed internal), ~37 KHz low consumption RC used as IWDG and/or RTC clock source.
4. HSE (high-speed external), 1 to 24 MHz crystal oscillator used directly or through the PLL as System clock source. Can be used also as RTC clock source.
5. LSE (low-speed external), 32 KHz oscillator used as RTC clock source.
6. PLL (clocked by HSI or HSE), featuring two different output clocks:
 - The first output is used to generate the high speed system clock (up to 32 MHz)

- The second output is used to generate the clock for the USB OTG FS (48 MHz)
- 7. CSS (Clock security system), once enable using the macro `__HAL_RCC_CSS_ENABLE()` and if a HSE clock failure occurs (HSE used directly or through PLL as System clock source), the System clock is automatically switched to MSI and an interrupt is generated if enabled. The interrupt is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector.
- 8. MCO1 (microcontroller clock output), used to output SYSCLK, HSI, LSI, MSI, LSE, HSE or PLL clock (through a configurable prescaler) on PA8 pin.

System, AHB and APB busses clocks configuration

1. Several clock sources can be used to drive the System clock (SYSCLK): MSI, HSI, HSE and PLL. The AHB clock (HCLK) is derived from System clock through configurable prescaler and used to clock the CPU, memory and peripherals mapped on AHB bus (DMA, GPIO...). APB1 (PCLK1) and APB2 (PCLK2) clocks are derived from AHB clock through configurable prescalers and used to clock the peripherals mapped on these busses. You can use `"HAL_RCC_GetSysClockFreq()"` function to retrieve the frequencies of these clocks. All the peripheral clocks are derived from the System clock (SYSCLK) except: RTC: RTC clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 16. You have to use `__HAL_RCC_RTC_CONFIG()` and `__HAL_RCC_RTC_ENABLE()` macros to configure this clock. LCD: LCD clock can be derived either from the LSI, LSE or HSE clock divided by 2 to 16. You have to use `__HAL_RCC_LCD_CONFIG()` macros to configure this clock. USB OTG FS and RTC: USB OTG FS require a frequency equal to 48 MHz to work correctly. This clock is derived of the main PLL through PLL Multiplier. IWDG clock which is always the LSI clock.
2. The maximum frequency of the SYSCLK and HCLK is 32 MHz, PCLK2 32 MHz and PCLK1 32 MHz. Depending on the device voltage range, the maximum frequency should be adapted accordingly (see [Table 21: "Number of wait states \(WS\) according to CPU clock \(HCLK\) frequency"](#)).
3. The following table gives the different clock source frequencies depending on the product voltage range (see [Table 22: "Clock frequency versus product voltage range"](#)).

Table 21: Number of wait states (WS) according to CPU clock (HCLK) frequency

Latency	HCLK clock frequency (MHz)		
	voltage range 1 (1.8 V)	voltage range 2 (1.5 V)	voltage range 3 (1.2 V)
0W(1CPU cycles)	$0 < \text{HCLK} \leq 16$	$0 < \text{HCLK} \leq 8$	$0 < \text{HCLK} \leq 2$
1WS(2CPU cycles)	$16 < \text{HCLK} \leq 32$	$8 < \text{HCLK} \leq 16$	$2 < \text{HCLK} \leq 4$

Table 22: Clock frequency versus product voltage range

Product voltage range	Clock frequency			
	MSI	HSI	HSE	PLL
Range 1 (1.8 V)	4.2 MHz	16 MHz	32 MHz HSE (external clock) or 24 MHz (crystal)	32 MHz (PLLVCO max = 96 MHz)
Range 2 (1.5 V)	4.2 MHz	16 MHz	16 MHz	16 MHz (PLLVCO max = 48 MHz)
Range 3 (1.2 V)	4.2 MHz	NA	8 MHz	4 MHz (PLLVCO max = 24 MHz)

This section contains the following APIs:

- [HAL_RCC_DeInit\(\)](#)

- [HAL_RCC_OscConfig\(\)](#)
- [HAL_RCC_ClockConfig\(\)](#)

33.3.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.

This section contains the following APIs:

- [HAL_RCC_MCOConfig\(\)](#)
- [HAL_RCC_EnableCSS\(\)](#)
- [HAL_RCC_DisableCSS\(\)](#)
- [HAL_RCC_GetSysClockFreq\(\)](#)
- [HAL_RCC_GetHCLKFreq\(\)](#)
- [HAL_RCC_GetPCLK1Freq\(\)](#)
- [HAL_RCC_GetPCLK2Freq\(\)](#)
- [HAL_RCC_GetOscConfig\(\)](#)
- [HAL_RCC_GetClockConfig\(\)](#)
- [HAL_RCC_NMI_IRQHandler\(\)](#)
- [HAL_RCC_CSSCallback\(\)](#)

33.3.5 HAL_RCC_DeInit

Function Name	void HAL_RCC_DeInit (void)
Function Description	Resets the RCC clock configuration to the default reset state.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • The default reset state of the clock configuration is given below: MSI ON and used as system clock source HSI, HSE and PLL OFF AHB, APB1 and APB2 prescaler set to 1. CSS and MCO1 OFF All interrupts disabled • This function doesn't modify the configuration of the Peripheral clocks LSI, LSE and RTC clocks

33.3.6 HAL_RCC_OscConfig

Function Name	HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)
Function Description	Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.
Parameters	<ul style="list-style-type: none"> • RCC_OscInitStruct: pointer to an RCC_OscInitTypeDef structure that contains the configuration information for the RCC Oscillators.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • The PLL is not disabled when used as system clock.

33.3.7 HAL_RCC_ClockConfig

Function Name	HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)
Function Description	Initializes the CPU, AHB and APB busses clocks according to the

specified parameters in the `RCC_ClkInitStruct`.

Parameters	<ul style="list-style-type: none"> • RCC_ClkInitStruct: pointer to an <code>RCC_OscInitTypeDef</code> structure that contains the configuration information for the RCC peripheral. • FLatency: FLASH Latency This parameter can be one of the following values: <code>FLASH_LATENCY_0</code>: FLASH 0 Latency cycle <code>FLASH_LATENCY_1</code>: FLASH 1 Latency cycle
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • The <code>SystemCoreClock</code> CMSIS variable is used to store System Clock Frequency and updated by <code>HAL_RCC_GetHCLKFreq()</code> function called within this function • The MSI is used (enabled by hardware) as system clock source after startup from Reset, wake-up from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). • A switch from one clock source to another occurs only if the target clock source is ready (clock stable after startup delay or PLL locked). If a clock source which is not yet ready is selected, the switch will occur when the clock source will be ready. You can use <code>HAL_RCC_GetClockConfig()</code> function to know which clock is currently used as system clock source. • Depending on the device voltage range, the software has to set correctly <code>HPRE[3:0]</code> bits to ensure that HCLK not exceed the maximum allowed frequency (for more details refer to section above "Initialization/de-initialization functions")

33.3.8 HAL_RCC_MCOConfig

Function Name	void HAL_RCC_MCOConfig (uint32_t RCC_MCOx, uint32_t RCC_MCOSource, uint32_t RCC_MCODiv)
Function Description	Selects the clock source to output on MCO pin.
Parameters	<ul style="list-style-type: none"> • RCC_MCOx: specifies the output direction for the clock source. This parameter can be one of the following values: <code>RCC_MCO</code>: Clock source to output on MCO1 pin(PA8). • RCC_MCOSource: specifies the clock source to output. This parameter can be one of the following values: <code>RCC_MCO1SOURCE_NOCLOCK</code>: No clock selected <code>RCC_MCO1SOURCE_SYSCLK</code>: System clock selected <code>RCC_MCO1SOURCE_HSI</code>: HSI oscillator clock selected <code>RCC_MCO1SOURCE_MSI</code>: MSI oscillator clock selected <code>RCC_MCO1SOURCE_HSE</code>: HSE oscillator clock selected <code>RCC_MCO1SOURCE_PLLCLK</code>: PLL clock selected <code>RCC_MCO1SOURCE_LSI</code>: LSI clock selected <code>RCC_MCO1SOURCE_LSE</code>: LSE clock selected • RCC_MCODiv: specifies the MCO DIV. This parameter can be one of the following values: <code>RCC_MCODIV_1</code>: no division applied to MCO clock <code>RCC_MCODIV_2</code>: division by 2 applied to MCO clock <code>RCC_MCODIV_4</code>: division by 4 applied to MCO clock <code>RCC_MCODIV_8</code>: division by 8 applied to MCO clock <code>RCC_MCODIV_16</code>: division by 16 applied to MCO clock

- | | |
|---------------|--|
| Return values | • None |
| Notes | • MCO pin should be configured in alternate function mode. |

33.3.9 HAL_RCC_EnableCSS

- | | |
|----------------------|---|
| Function Name | void HAL_RCC_EnableCSS (void) |
| Function Description | Enables the Clock Security System. |
| Return values | • None |
| Notes | • If a failure is detected on the HSE oscillator clock, this oscillator is automatically disabled and an interrupt is generated to inform the software about the failure (Clock Security System Interrupt, CSSI), allowing the MCU to perform rescue operations. The CSSI is linked to the Cortex-M3 NMI (Non-Maskable Interrupt) exception vector. |

33.3.10 HAL_RCC_DisableCSS

- | | |
|----------------------|--|
| Function Name | void HAL_RCC_DisableCSS (void) |
| Function Description | Disables the Clock Security System. |
| Return values | • None |

33.3.11 HAL_RCC_GetSysClockFreq

- | | |
|----------------------|--|
| Function Name | uint32_t HAL_RCC_GetSysClockFreq (void) |
| Function Description | Returns the SYSCLK frequency. |
| Return values | • SYSCLK frequency |
| Notes | <ul style="list-style-type: none">• The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:• If SYSCLK source is MSI, function returns values based on MSI Value as defined by the MSI range.• If SYSCLK source is HSI, function returns values based on HSI_VALUE(*)• If SYSCLK source is HSE, function returns values based on HSE_VALUE(**)• If SYSCLK source is PLL, function returns values based on HSE_VALUE(**) or HSI_VALUE(*) multiplied/divided by the PLL factors.• (*) HSI_VALUE is a constant defined in stm32l1xx_hal_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.• (**) HSE_VALUE is a constant defined in stm32l1xx_hal_conf.h file (default value 8 MHz), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.• The result of this function could be not correct when using fractional value for HSE crystal.• This function can be used by the user application to compute the baudrate for the communication peripherals or configure other parameters. |

- Each time SYSCLK changes, this function must be called to update the right SYSCLK value. Otherwise, any configuration based on this function will be incorrect.

33.3.12 HAL_RCC_GetHCLKFreq

Function Name	uint32_t HAL_RCC_GetHCLKFreq (void)
Function Description	Returns the HCLK frequency.
Return values	<ul style="list-style-type: none"> • HCLK frequency
Notes	<ul style="list-style-type: none"> • Each time HCLK changes, this function must be called to update the right HCLK value. Otherwise, any configuration based on this function will be incorrect. • The SystemCoreClock CMSIS variable is used to store System Clock Frequency and updated within this function

33.3.13 HAL_RCC_GetPCLK1Freq

Function Name	uint32_t HAL_RCC_GetPCLK1Freq (void)
Function Description	Returns the PCLK1 frequency.
Return values	<ul style="list-style-type: none"> • PCLK1 frequency
Notes	<ul style="list-style-type: none"> • Each time PCLK1 changes, this function must be called to update the right PCLK1 value. Otherwise, any configuration based on this function will be incorrect.

33.3.14 HAL_RCC_GetPCLK2Freq

Function Name	uint32_t HAL_RCC_GetPCLK2Freq (void)
Function Description	Returns the PCLK2 frequency.
Return values	<ul style="list-style-type: none"> • PCLK2 frequency
Notes	<ul style="list-style-type: none"> • Each time PCLK2 changes, this function must be called to update the right PCLK2 value. Otherwise, any configuration based on this function will be incorrect.

33.3.15 HAL_RCC_GetOscConfig

Function Name	void HAL_RCC_GetOscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)
Function Description	Configures the RCC_OscInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> • RCC_OscInitStruct: pointer to an RCC_OscInitTypeDef structure that will be configured.
Return values	<ul style="list-style-type: none"> • None

33.3.16 HAL_RCC_GetClockConfig

Function Name	void HAL_RCC_GetClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t * pFLatency)
---------------	---

Function Description	Get the RCC_ClkInitStruct according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> • RCC_ClkInitStruct: pointer to an RCC_ClkInitTypeDef structure that contains the current clock configuration. • pFLatency: Pointer on the Flash Latency.
Return values	<ul style="list-style-type: none"> • None

33.3.17 HAL_RCC_NMI_IRQHandler

Function Name	void HAL_RCC_NMI_IRQHandler (void)
Function Description	This function handles the RCC CSS interrupt request.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • This API should be called under the NMI_Handler().

33.3.18 HAL_RCC_CSSCallback

Function Name	void HAL_RCC_CSSCallback (void)
Function Description	RCC Clock Security System interrupt callback.
Return values	<ul style="list-style-type: none"> • none

33.4 RCC Firmware driver defines

33.4.1 RCC

AHB Peripheral Clock Sleep Enable Disable Status

```

__HAL_RCC_GPIOA_IS_CLK_SLEEP_ENABLED
__HAL_RCC_GPIOB_IS_CLK_SLEEP_ENABLED
__HAL_RCC_GPIOC_IS_CLK_SLEEP_ENABLED
__HAL_RCC_GPIOD_IS_CLK_SLEEP_ENABLED
__HAL_RCC_GPIOH_IS_CLK_SLEEP_ENABLED
__HAL_RCC_CRC_IS_CLK_SLEEP_ENABLED
__HAL_RCC_FLITF_IS_CLK_SLEEP_ENABLED
__HAL_RCC_DMA1_IS_CLK_SLEEP_ENABLED
__HAL_RCC_GPIOA_IS_CLK_SLEEP_DISABLED
__HAL_RCC_GPIOB_IS_CLK_SLEEP_DISABLED
__HAL_RCC_GPIOC_IS_CLK_SLEEP_DISABLED
__HAL_RCC_GPIOD_IS_CLK_SLEEP_DISABLED
__HAL_RCC_GPIOH_IS_CLK_SLEEP_DISABLED
__HAL_RCC_CRC_IS_CLK_SLEEP_DISABLED
__HAL_RCC_FLITF_IS_CLK_SLEEP_DISABLED
__HAL_RCC_DMA1_IS_CLK_SLEEP_DISABLED

```

AHB Clock Source

RCC_SYSCLK_DIV1
RCC_SYSCLK_DIV2
RCC_SYSCLK_DIV4
RCC_SYSCLK_DIV8
RCC_SYSCLK_DIV16
RCC_SYSCLK_DIV64
RCC_SYSCLK_DIV128
RCC_SYSCLK_DIV256
RCC_SYSCLK_DIV512

AHB Peripheral Clock Enable Disable Status

__HAL_RCC_GPIOA_IS_CLK_ENABLED
__HAL_RCC_GPIOB_IS_CLK_ENABLED
__HAL_RCC_GPIOC_IS_CLK_ENABLED
__HAL_RCC_GPIOD_IS_CLK_ENABLED
__HAL_RCC_GPIOH_IS_CLK_ENABLED
__HAL_RCC_CRC_IS_CLK_ENABLED
__HAL_RCC_FLITF_IS_CLK_ENABLED
__HAL_RCC_DMA1_IS_CLK_ENABLED
__HAL_RCC_GPIOA_IS_CLK_DISABLED
__HAL_RCC_GPIOB_IS_CLK_DISABLED
__HAL_RCC_GPIOC_IS_CLK_DISABLED
__HAL_RCC_GPIOD_IS_CLK_DISABLED
__HAL_RCC_GPIOH_IS_CLK_DISABLED
__HAL_RCC_CRC_IS_CLK_DISABLED
__HAL_RCC_FLITF_IS_CLK_DISABLED
__HAL_RCC_DMA1_IS_CLK_DISABLED

APB1 APB2 Clock Source

RCC_HCLK_DIV1
RCC_HCLK_DIV2
RCC_HCLK_DIV4
RCC_HCLK_DIV8
RCC_HCLK_DIV16

APB1 Clock Enable Disable

__HAL_RCC_TIM2_CLK_ENABLE
__HAL_RCC_TIM3_CLK_ENABLE
__HAL_RCC_TIM4_CLK_ENABLE

__HAL_RCC_TIM6_CLK_ENABLE
__HAL_RCC_TIM7_CLK_ENABLE
__HAL_RCC_WWDG_CLK_ENABLE
__HAL_RCC_SPI2_CLK_ENABLE
__HAL_RCC_USART2_CLK_ENABLE
__HAL_RCC_USART3_CLK_ENABLE
__HAL_RCC_I2C1_CLK_ENABLE
__HAL_RCC_I2C2_CLK_ENABLE
__HAL_RCC_USB_CLK_ENABLE
__HAL_RCC_PWR_CLK_ENABLE
__HAL_RCC_DAC_CLK_ENABLE
__HAL_RCC_COMP_CLK_ENABLE
__HAL_RCC_TIM2_CLK_DISABLE
__HAL_RCC_TIM3_CLK_DISABLE
__HAL_RCC_TIM4_CLK_DISABLE
__HAL_RCC_TIM6_CLK_DISABLE
__HAL_RCC_TIM7_CLK_DISABLE
__HAL_RCC_WWDG_CLK_DISABLE
__HAL_RCC_SPI2_CLK_DISABLE
__HAL_RCC_USART2_CLK_DISABLE
__HAL_RCC_USART3_CLK_DISABLE
__HAL_RCC_I2C1_CLK_DISABLE
__HAL_RCC_I2C2_CLK_DISABLE
__HAL_RCC_USB_CLK_DISABLE
__HAL_RCC_PWR_CLK_DISABLE
__HAL_RCC_DAC_CLK_DISABLE
__HAL_RCC_COMP_CLK_DISABLE

APB1 Peripheral Clock Sleep Enable Disable Status

__HAL_RCC_TIM2_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM3_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM4_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM6_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM7_IS_CLK_SLEEP_ENABLED
__HAL_RCC_WWDG_IS_CLK_SLEEP_ENABLED
__HAL_RCC_SPI2_IS_CLK_SLEEP_ENABLED
__HAL_RCC_USART2_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART3_IS_CLK_SLEEP_ENABLED
__HAL_RCC_I2C1_IS_CLK_SLEEP_ENABLED
__HAL_RCC_I2C2_IS_CLK_SLEEP_ENABLED
__HAL_RCC_USB_IS_CLK_SLEEP_ENABLED
__HAL_RCC_PWR_IS_CLK_SLEEP_ENABLED
__HAL_RCC_DAC_IS_CLK_SLEEP_ENABLED
__HAL_RCC_COMP_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM2_IS_CLK_SLEEP_DISABLED
__HAL_RCC_TIM3_IS_CLK_SLEEP_DISABLED
__HAL_RCC_TIM4_IS_CLK_SLEEP_DISABLED
__HAL_RCC_TIM6_IS_CLK_SLEEP_DISABLED
__HAL_RCC_TIM7_IS_CLK_SLEEP_DISABLED
__HAL_RCC_WWDG_IS_CLK_SLEEP_DISABLED
__HAL_RCC_SPI2_IS_CLK_SLEEP_DISABLED
__HAL_RCC_USART2_IS_CLK_SLEEP_DISABLED
__HAL_RCC_USART3_IS_CLK_SLEEP_DISABLED
__HAL_RCC_I2C1_IS_CLK_SLEEP_DISABLED
__HAL_RCC_I2C2_IS_CLK_SLEEP_DISABLED
__HAL_RCC_USB_IS_CLK_SLEEP_DISABLED
__HAL_RCC_PWR_IS_CLK_SLEEP_DISABLED
__HAL_RCC_DAC_IS_CLK_SLEEP_DISABLED
__HAL_RCC_COMP_IS_CLK_SLEEP_DISABLED

APB1 Force Release Reset

__HAL_RCC_APB1_FORCE_RESET
__HAL_RCC_TIM2_FORCE_RESET
__HAL_RCC_TIM3_FORCE_RESET
__HAL_RCC_TIM4_FORCE_RESET
__HAL_RCC_TIM6_FORCE_RESET
__HAL_RCC_TIM7_FORCE_RESET
__HAL_RCC_WWDG_FORCE_RESET
__HAL_RCC_SPI2_FORCE_RESET
__HAL_RCC_USART2_FORCE_RESET
__HAL_RCC_USART3_FORCE_RESET
__HAL_RCC_I2C1_FORCE_RESET
__HAL_RCC_I2C2_FORCE_RESET
__HAL_RCC_USB_FORCE_RESET

__HAL_RCC_PWR_FORCE_RESET
__HAL_RCC_DAC_FORCE_RESET
__HAL_RCC_COMP_FORCE_RESET
__HAL_RCC_APB1_RELEASE_RESET
__HAL_RCC_TIM2_RELEASE_RESET
__HAL_RCC_TIM3_RELEASE_RESET
__HAL_RCC_TIM4_RELEASE_RESET
__HAL_RCC_TIM6_RELEASE_RESET
__HAL_RCC_TIM7_RELEASE_RESET
__HAL_RCC_WWDG_RELEASE_RESET
__HAL_RCC_SPI2_RELEASE_RESET
__HAL_RCC_USART2_RELEASE_RESET
__HAL_RCC_USART3_RELEASE_RESET
__HAL_RCC_I2C1_RELEASE_RESET
__HAL_RCC_I2C2_RELEASE_RESET
__HAL_RCC_USB_RELEASE_RESET
__HAL_RCC_PWR_RELEASE_RESET
__HAL_RCC_DAC_RELEASE_RESET
__HAL_RCC_COMP_RELEASE_RESET

APB1 Peripheral Clock Enable Disable Status

__HAL_RCC_TIM2_IS_CLK_ENABLED
__HAL_RCC_TIM3_IS_CLK_ENABLED
__HAL_RCC_TIM4_IS_CLK_ENABLED
__HAL_RCC_TIM6_IS_CLK_ENABLED
__HAL_RCC_TIM7_IS_CLK_ENABLED
__HAL_RCC_WWDG_IS_CLK_ENABLED
__HAL_RCC_SPI2_IS_CLK_ENABLED
__HAL_RCC_USART2_IS_CLK_ENABLED
__HAL_RCC_USART3_IS_CLK_ENABLED
__HAL_RCC_I2C1_IS_CLK_ENABLED
__HAL_RCC_I2C2_IS_CLK_ENABLED
__HAL_RCC_USB_IS_CLK_ENABLED
__HAL_RCC_PWR_IS_CLK_ENABLED
__HAL_RCC_DAC_IS_CLK_ENABLED
__HAL_RCC_COMP_IS_CLK_ENABLED
__HAL_RCC_TIM2_IS_CLK_DISABLED

__HAL_RCC_TIM3_IS_CLK_DISABLED
__HAL_RCC_TIM4_IS_CLK_DISABLED
__HAL_RCC_TIM6_IS_CLK_DISABLED
__HAL_RCC_TIM7_IS_CLK_DISABLED
__HAL_RCC_WWDG_IS_CLK_DISABLED
__HAL_RCC_SPI2_IS_CLK_DISABLED
__HAL_RCC_USART2_IS_CLK_DISABLED
__HAL_RCC_USART3_IS_CLK_DISABLED
__HAL_RCC_I2C1_IS_CLK_DISABLED
__HAL_RCC_I2C2_IS_CLK_DISABLED
__HAL_RCC_USB_IS_CLK_DISABLED
__HAL_RCC_PWR_IS_CLK_DISABLED
__HAL_RCC_DAC_IS_CLK_DISABLED
__HAL_RCC_COMP_IS_CLK_DISABLED

APB2 Clock Enable Disable

__HAL_RCC_SYSCFG_CLK_ENABLE
__HAL_RCC_TIM9_CLK_ENABLE
__HAL_RCC_TIM10_CLK_ENABLE
__HAL_RCC_TIM11_CLK_ENABLE
__HAL_RCC_ADC1_CLK_ENABLE
__HAL_RCC_SPI1_CLK_ENABLE
__HAL_RCC_USART1_CLK_ENABLE
__HAL_RCC_SYSCFG_CLK_DISABLE
__HAL_RCC_TIM9_CLK_DISABLE
__HAL_RCC_TIM10_CLK_DISABLE
__HAL_RCC_TIM11_CLK_DISABLE
__HAL_RCC_ADC1_CLK_DISABLE
__HAL_RCC_SPI1_CLK_DISABLE
__HAL_RCC_USART1_CLK_DISABLE

APB2 Peripheral Clock Sleep Enable Disable Status

__HAL_RCC_SYSCFG_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM9_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM10_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM11_IS_CLK_SLEEP_ENABLED
__HAL_RCC_ADC1_IS_CLK_SLEEP_ENABLED
__HAL_RCC_SPI1_IS_CLK_SLEEP_ENABLED

__HAL_RCC_USART1_IS_CLK_SLEEP_ENABLED
__HAL_RCC_SYSCFG_IS_CLK_SLEEP_DISABLED
__HAL_RCC_TIM9_IS_CLK_SLEEP_DISABLED
__HAL_RCC_TIM10_IS_CLK_SLEEP_DISABLED
__HAL_RCC_TIM11_IS_CLK_SLEEP_DISABLED
__HAL_RCC_ADC1_IS_CLK_SLEEP_DISABLED
__HAL_RCC_SPI1_IS_CLK_SLEEP_DISABLED
__HAL_RCC_USART1_IS_CLK_SLEEP_DISABLED

APB2 Force Release Reset

__HAL_RCC_APB2_FORCE_RESET
__HAL_RCC_SYSCFG_FORCE_RESET
__HAL_RCC_TIM9_FORCE_RESET
__HAL_RCC_TIM10_FORCE_RESET
__HAL_RCC_TIM11_FORCE_RESET
__HAL_RCC_ADC1_FORCE_RESET
__HAL_RCC_SPI1_FORCE_RESET
__HAL_RCC_USART1_FORCE_RESET
__HAL_RCC_APB2_RELEASE_RESET
__HAL_RCC_SYSCFG_RELEASE_RESET
__HAL_RCC_TIM9_RELEASE_RESET
__HAL_RCC_TIM10_RELEASE_RESET
__HAL_RCC_TIM11_RELEASE_RESET
__HAL_RCC_ADC1_RELEASE_RESET
__HAL_RCC_SPI1_RELEASE_RESET
__HAL_RCC_USART1_RELEASE_RESET

APB2 Peripheral Clock Enable Disable Status

__HAL_RCC_SYSCFG_IS_CLK_ENABLED
__HAL_RCC_TIM9_IS_CLK_ENABLED
__HAL_RCC_TIM10_IS_CLK_ENABLED
__HAL_RCC_TIM11_IS_CLK_ENABLED
__HAL_RCC_ADC1_IS_CLK_ENABLED
__HAL_RCC_SPI1_IS_CLK_ENABLED
__HAL_RCC_USART1_IS_CLK_ENABLED
__HAL_RCC_SYSCFG_IS_CLK_DISABLED
__HAL_RCC_TIM9_IS_CLK_DISABLED
__HAL_RCC_TIM10_IS_CLK_DISABLED

__HAL_RCC_TIM11_IS_CLK_DISABLED
__HAL_RCC_ADC1_IS_CLK_DISABLED
__HAL_RCC_SPI1_IS_CLK_DISABLED
__HAL_RCC_USART1_IS_CLK_DISABLED

BitAddress AliasRegion

RCC_CR_OFFSET_BB
RCC_CFGR_OFFSET_BB
RCC_CIR_OFFSET_BB
RCC_CSR_OFFSET_BB
HSION_BITNUMBER
RCC_CR_HSION_BB
MSION_BITNUMBER
RCC_CR_MSION_BB
HSEON_BITNUMBER
CR_HSEON_BB
CSSON_BITNUMBER
RCC_CR_CSSON_BB
PLLON_BITNUMBER
RCC_CR_PLLON_BB
LSION_BITNUMBER
RCC_CSR_LSION_BB
LSEON_BITNUMBER
RCC_CSR_LSEON_BB
LSEBYP_BITNUMBER
RCC_CSR_LSEBYP_BB
RTCEN_BITNUMBER
RCC_CSR_RTCEN_BB
RTCRST_BITNUMBER
RCC_CSR_RTCRST_BB

Flags

RCC_FLAG_HSIRDY	Internal High Speed clock ready flag
RCC_FLAG_MSIRDY	MSI clock ready flag
RCC_FLAG_HSERDY	External High Speed clock ready flag
RCC_FLAG_PLLRDY	PLL clock ready flag
RCC_FLAG_LSIRDY	Internal Low Speed oscillator Ready
RCC_FLAG_LSECSS	CSS on LSE failure Detection

RCC_FLAG_RMV	Remove reset flag
RCC_FLAG_OBLRST	Options bytes loading reset flag
RCC_FLAG_PINRST	PIN reset flag
RCC_FLAG_PORRST	POR/PDR reset flag
RCC_FLAG_SFTRST	Software Reset flag
RCC_FLAG_IWDGRST	Independent Watchdog reset flag
RCC_FLAG_WWDGRST	Window watchdog reset flag
RCC_FLAG_LPWRST	Low-Power reset flag
RCC_FLAG_LSERDY	External Low Speed oscillator Ready

Flags Interrupts Management

`__HAL_RCC_ENABLE_IT`

Description:

- Enable RCC interrupt.

Parameters:

- `__INTERERRUPT__`: specifies the RCC interrupt sources to be enabled. This parameter can be any combination of the following values:
 - RCC_IT_LSIRDY: LSI ready interrupt
 - RCC_IT_LSERDY: LSE ready interrupt
 - RCC_IT_HSIRDY: HSI ready interrupt
 - RCC_IT_HSERDY: HSE ready interrupt
 - RCC_IT_PLLRDY: main PLL ready interrupt
 - RCC_IT_MSIRDY: MSI ready interrupt
 - RCC_IT_LSECSS: LSE CSS interrupt (not available for STM32L100xB || STM32L151xB || STM32L152xB devices)

`__HAL_RCC_DISABLE_IT`

Description:

- Disable RCC interrupt.

Parameters:

- `__INTERERRUPT__`: specifies the RCC interrupt sources to be disabled. This parameter can be any combination of the following values:
 - RCC_IT_LSIRDY: LSI ready interrupt
 - RCC_IT_LSERDY: LSE ready interrupt
 - RCC_IT_HSIRDY: HSI ready interrupt
 - RCC_IT_HSERDY: HSE ready interrupt
 - RCC_IT_PLLRDY: main PLL ready interrupt
 - RCC_IT_MSIRDY: MSI ready interrupt
 - RCC_IT_LSECSS: LSE CSS interrupt (not available for STM32L100xB ||

STM32L151xB || STM32L152xB
devices)

__HAL_RCC_CLEAR_IT

Description:

- Clear the RCC's interrupt pending bits.

Parameters:

- **__INTERRUPT__**: specifies the interrupt pending bit to clear. This parameter can be any combination of the following values:
 - RCC_IT_LSIRDY: LSI ready interrupt.
 - RCC_IT_LSERDY: LSE ready interrupt.
 - RCC_IT_HSIRDY: HSI ready interrupt.
 - RCC_IT_HSERDY: HSE ready interrupt.
 - RCC_IT_PLLRDY: Main PLL ready interrupt.
 - RCC_IT_MSIRDY: MSI ready interrupt
 - RCC_IT_LSECSS: LSE CSS interrupt (not available for STM32L100xB || STM32L151xB || STM32L152xB devices)
 - RCC_IT_CSS: Clock Security System interrupt

__HAL_RCC_GET_IT

Description:

- Check the RCC's interrupt has occurred or not.

Parameters:

- **__INTERRUPT__**: specifies the RCC interrupt source to check. This parameter can be one of the following values:
 - RCC_IT_LSIRDY: LSI ready interrupt.
 - RCC_IT_LSERDY: LSE ready interrupt.
 - RCC_IT_HSIRDY: HSI ready interrupt.
 - RCC_IT_HSERDY: HSE ready interrupt.
 - RCC_IT_PLLRDY: Main PLL ready interrupt.
 - RCC_IT_MSIRDY: MSI ready interrupt
 - RCC_IT_LSECSS: LSE CSS interrupt (not available for STM32L100xB || STM32L151xB || STM32L152xB devices)
 - RCC_IT_CSS: Clock Security System interrupt

Return value:

- The: new state of **__INTERRUPT__** (TRUE or FALSE).

__HAL_RCC_CLEAR_RESET_FLAGS

The reset flags are: RCC_FLAG_PINRST, RCC_FLAG_PORRST, RCC_FLAG_SFTRST, RCC_FLAG_IWDGRST,

__HAL_RCC_GET_FLAGRCC_FLAG_WWDGRST,
RCC_FLAG_LPWRST**Description:**

- Check RCC flag is set or not.

Parameters:

- **__FLAG__**: specifies the flag to check. This parameter can be one of the following values:
 - RCC_FLAG_HSIIRDY: HSI oscillator clock ready.
 - RCC_FLAG_MSIRDY: MSI oscillator clock ready.
 - RCC_FLAG_HSERDY: HSE oscillator clock ready.
 - RCC_FLAG_PLLRDY: Main PLL clock ready.
 - RCC_FLAG_LSERDY: LSE oscillator clock ready.
 - RCC_FLAG_LSECSS: CSS on LSE failure Detection (*)
 - RCC_FLAG_LSIRDY: LSI oscillator clock ready.
 - RCC_FLAG_BORRST: POR/PDR or BOR reset.
 - RCC_FLAG_PINRST: Pin reset.
 - RCC_FLAG_PORRST: POR/PDR reset.
 - RCC_FLAG_SFTRST: Software reset.
 - RCC_FLAG_IWDGRST: Independent Watchdog reset.
 - RCC_FLAG_WWDGRST: Window Watchdog reset.
 - RCC_FLAG_LPWRST: Low Power reset.

Return value:

- The: new state of **__FLAG__** (TRUE or FALSE).

Notes:

- (*) This bit is available in high and medium+ density devices only.

Get Clock source**__HAL_RCC_SYSCLK_CONFIG****Description:**

- Macro to configure the system clock source.

Parameters:

- **__RCC_SYSCLKSOURCE__**: specifies the system clock source. This parameter can be one of the following values:
 - RCC_SYSCLKSOURCE_MSI: MSI

- oscillator is used as system clock source.
- RCC_SYSCLKSOURCE_HSI: HSI oscillator is used as system clock source.
- RCC_SYSCLKSOURCE_HSE: HSE oscillator is used as system clock source.
- RCC_SYSCLKSOURCE_PLLCLK: PLL output is used as system clock source.

`__HAL_RCC_GET_SYSCLK_SOURCE`

Description:

- Macro to get the clock source used as system clock.

Return value:

- The: clock source used as system clock. The returned value can be one of the following:
 - RCC_SYSCLKSOURCE_STATUS_MSI: MSI used as system clock
 - RCC_SYSCLKSOURCE_STATUS_HSI: HSI used as system clock
 - RCC_SYSCLKSOURCE_STATUS_HSE: HSE used as system clock
 - RCC_SYSCLKSOURCE_STATUS_PLLCLK: PLL used as system clock

HSE Config

RCC_HSE_OFF HSE clock deactivation

RCC_HSE_ON HSE clock activation

RCC_HSE_BYPASS External clock source for HSE clock

HSE Configuration

`__HAL_RCC_HSE_CONFIG`

Description:

- Macro to configure the External High Speed oscillator (HSE).

Parameters:

- `__STATE__`: specifies the new state of the HSE. This parameter can be one of the following values:
 - RCC_HSE_OFF: turn OFF the HSE oscillator, HSERDY flag goes low after 6 HSE oscillator clock cycles.
 - RCC_HSE_ON: turn ON the HSE oscillator
 - RCC_HSE_BYPASS: HSE oscillator bypassed with external clock

Notes:

- After enabling the HSE (RCC_HSE_ON or RCC_HSE_Bypass), the application software should wait on HSERDY flag to be set indicating that HSE clock is stable and can be used to clock the PLL and/or system clock. HSE state can not be changed if it is used directly or through the PLL as system clock. In this case, you have to select another source of the system clock then change the HSE state (ex. disable

it). The HSE is stopped by hardware when entering STOP and STANDBY modes. This function reset the CSSON bit, so if the Clock security system(CSS) was previously enabled you have to enable it again after calling this function.

HSI Config

RCC_HSI_OFF

HSI clock deactivation

RCC_HSI_ON

HSI clock activation

RCC_HSICALIBRATION_DEFAULT

HSI Configuration

__HAL_RCC_HSI_ENABLE

Notes:

- The HSI is stopped by hardware when entering STOP and STANDBY modes. HSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the HSI. After enabling the HSI, the application software should wait on HSIRDY flag to be set indicating that HSI clock is stable and can be used as system clock source. When the HSI is stopped, HSIRDY flag goes low after 6 HSI oscillator clock cycles.

__HAL_RCC_HSI_DISABLE

__HAL_RCC_HSI_CALIBRATIONVALUE_ADJUST

Description:

- macro to adjust the Internal High Speed oscillator (HSI) calibration value.

Parameters:

- `_HSICALIBRATIONVALUE_`: specifies the calibration trimming value. (default is `RCC_HSICALIBRATION_DEFAULT`). This parameter must be a number between 0 and 0x1F.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal HSI RC.

Interrupts

RCC_IT_LSIRDY	LSI Ready Interrupt flag
RCC_IT_LSERDY	LSE Ready Interrupt flag
RCC_IT_HSIRDY	HSI Ready Interrupt flag
RCC_IT_HSERDY	HSE Ready Interrupt flag
RCC_IT_PLLRDY	PLL Ready Interrupt flag
RCC_IT_MSIRDY	MSI Ready Interrupt flag
RCC_IT_LSECSS	LSE Clock Security System Interrupt flag
RCC_IT_CSS	Clock Security System Interrupt flag

LSE Config

RCC_LSE_OFF	LSE clock deactivation
RCC_LSE_ON	LSE clock activation
RCC_LSE_BYPASS	External clock source for LSE clock

LSE Configuration

__HAL_RCC_LSE_CONFIG **Description:**

- Macro to configure the External Low Speed oscillator (LSE).

Parameters:

- __STATE__**: specifies the new state of the LSE. This parameter can be one of the following values:
 - RCC_LSE_OFF: turn OFF the LSE oscillator, LSERDY flag goes low after 6 LSE oscillator clock cycles.
 - RCC_LSE_ON: turn ON the LSE oscillator.
 - RCC_LSE_BYPASS: LSE oscillator bypassed with external clock.

Notes:

- Transitions LSE Bypass to LSE On and LSE On to LSE Bypass are not supported by this macro. As the LSE is in the Backup domain and write access is denied to this domain after reset, you have to enable write access using HAL_PWR_EnableBkUpAccess() function before to configure the LSE (to be done once after reset). After enabling the LSE (RCC_LSE_ON or RCC_LSE_BYPASS), the application software should wait on LSERDY flag to be set indicating that LSE clock is stable and can be used to clock the RTC.

LSI Config

RCC_LSI_OFF	LSI clock deactivation
RCC_LSI_ON	LSI clock activation

LSI Configuration

__HAL_RCC_LSI_ENABLE **Notes:**

- After enabling the LSI, the application software should wait on LSIRDY flag to be set indicating that LSI clock

is stable and can be used to clock the IWDG and/or the RTC. LSI can not be disabled if the IWDG is running. When the LSI is stopped, LSIRDY flag goes low after 6 LSI oscillator clock cycles.

__HAL_RCC_LSI_DISABLE

MCO1 Clock Source

RCC_MCO1SOURCE_NOCLOCK

RCC_MCO1SOURCE_SYSCLK

RCC_MCO1SOURCE_MSI

RCC_MCO1SOURCE_HSI

RCC_MCO1SOURCE_LSE

RCC_MCO1SOURCE_LSI

RCC_MCO1SOURCE_HSE

RCC_MCO1SOURCE_PLLCLK

MCO Clock Prescaler

RCC_MCODIV_1

RCC_MCODIV_2

RCC_MCODIV_4

RCC_MCODIV_8

RCC_MCODIV_16

MCO Index

RCC_MCO1

RCC_MCO MCO1 to be compliant with other families with 2 MCOs

MSI Clock Range

RCC_MSIRANGE_0 MSI = 65.536 KHz

RCC_MSIRANGE_1 MSI = 131.072 KHz

RCC_MSIRANGE_2 MSI = 262.144 KHz

RCC_MSIRANGE_3 MSI = 524.288 KHz

RCC_MSIRANGE_4 MSI = 1.048 MHz

RCC_MSIRANGE_5 MSI = 2.097 MHz

RCC_MSIRANGE_6 MSI = 4.194 MHz

MSI Config

RCC_MSI_OFF

RCC_MSI_ON

RCC_MSICALIBRATION_DEFAULT

MSI Configuration

__HAL_RCC_MSI_ENABLE

Notes:

- After enabling the MSI, the

application software should wait on MSIRDY flag to be set indicating that MSI clock is stable and can be used as system clock source.

__HAL_RCC_MSI_DISABLE

Notes:

- The MSI is stopped by hardware when entering STOP and STANDBY modes. It is used (enabled by hardware) as system clock source after startup from Reset, wakeup from STOP and STANDBY mode, or in case of failure of the HSE used directly or indirectly as system clock (if the Clock Security System CSS is enabled). MSI can not be stopped if it is used as system clock source. In this case, you have to select another source of the system clock then stop the MSI. When the MSI is stopped, MSIRDY flag goes low after 6 MSI oscillator clock cycles.

__HAL_RCC_MSI_CALIBRATIONVALUE_ADJUST

Description:

- Macro adjusts Internal Multi Speed oscillator (MSI) calibration value.

Parameters:

- `_MSICALIBRATIONVALUE_`: specifies the calibration trimming value. (default is `RCC_MSICALIBRATION_DEFAULT`). This parameter must be a number between 0 and 0xFF.

Notes:

- The calibration is used to compensate for the variations in voltage and temperature that influence the frequency of the internal MSI RC.

__HAL_RCC_MSI_RANGE_CONFIG

__HAL_RCC_GET_MSI_RANGE

Description:

- Macro to get the Internal Multi Speed oscillator (MSI) clock range in run mode.

Return value:

- MSI: clock range. This parameter must be one of the following values:
 - RCC_MSIRANGE_0: MSI clock is around 65.536 KHz
 - RCC_MSIRANGE_1: MSI clock is around 131.072 KHz
 - RCC_MSIRANGE_2: MSI clock is around 262.144 KHz
 - RCC_MSIRANGE_3: MSI clock is around 524.288 KHz
 - RCC_MSIRANGE_4: MSI clock is around 1.048 MHz
 - RCC_MSIRANGE_5: MSI clock is around 2.097 MHz (default after Reset or wake-up from STANDBY)
 - RCC_MSIRANGE_6: MSI clock is around 4.194 MHz

Oscillator Type`RCC_OSCILLATORTYPE_NONE``RCC_OSCILLATORTYPE_HSE``RCC_OSCILLATORTYPE_HSI``RCC_OSCILLATORTYPE_LSE``RCC_OSCILLATORTYPE_LSI``RCC_OSCILLATORTYPE_MSI`***Peripheral Clock Enable Disable***`__HAL_RCC_GPIOA_CLK_ENABLE``__HAL_RCC_GPIOB_CLK_ENABLE``__HAL_RCC_GPIOC_CLK_ENABLE``__HAL_RCC_GPIOD_CLK_ENABLE``__HAL_RCC_GPIOH_CLK_ENABLE``__HAL_RCC_CRC_CLK_ENABLE``__HAL_RCC_FLITF_CLK_ENABLE``__HAL_RCC_DMA1_CLK_ENABLE``__HAL_RCC_GPIOA_CLK_DISABLE``__HAL_RCC_GPIOB_CLK_DISABLE``__HAL_RCC_GPIOC_CLK_DISABLE``__HAL_RCC_GPIOD_CLK_DISABLE``__HAL_RCC_GPIOH_CLK_DISABLE`

__HAL_RCC_CRC_CLK_DISABLE
__HAL_RCC_FLITF_CLK_DISABLE
__HAL_RCC_DMA1_CLK_DISABLE

RCC Peripheral Clock Force Release

__HAL_RCC_AHB_FORCE_RESET
__HAL_RCC_GPIOA_FORCE_RESET
__HAL_RCC_GPIOB_FORCE_RESET
__HAL_RCC_GPIOC_FORCE_RESET
__HAL_RCC_GPIOD_FORCE_RESET
__HAL_RCC_GPIOH_FORCE_RESET
__HAL_RCC_CRC_FORCE_RESET
__HAL_RCC_FLITF_FORCE_RESET
__HAL_RCC_DMA1_FORCE_RESET
__HAL_RCC_AHB_RELEASE_RESET
__HAL_RCC_GPIOA_RELEASE_RESET
__HAL_RCC_GPIOB_RELEASE_RESET
__HAL_RCC_GPIOC_RELEASE_RESET
__HAL_RCC_GPIOD_RELEASE_RESET
__HAL_RCC_GPIOH_RELEASE_RESET
__HAL_RCC_CRC_RELEASE_RESET
__HAL_RCC_FLITF_RELEASE_RESET
__HAL_RCC_DMA1_RELEASE_RESET

RCC Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPIOA_CLK_SLEEP_ENABLE
__HAL_RCC_GPIOB_CLK_SLEEP_ENABLE
__HAL_RCC_GPIOC_CLK_SLEEP_ENABLE
__HAL_RCC_GPIOD_CLK_SLEEP_ENABLE
__HAL_RCC_GPIOH_CLK_SLEEP_ENABLE
__HAL_RCC_CRC_CLK_SLEEP_ENABLE
__HAL_RCC_FLITF_CLK_SLEEP_ENABLE
__HAL_RCC_DMA1_CLK_SLEEP_ENABLE
__HAL_RCC_GPIOA_CLK_SLEEP_DISABLE
__HAL_RCC_GPIOB_CLK_SLEEP_DISABLE
__HAL_RCC_GPIOC_CLK_SLEEP_DISABLE
__HAL_RCC_GPIOD_CLK_SLEEP_DISABLE
__HAL_RCC_GPIOH_CLK_SLEEP_DISABLE

__HAL_RCC_CRC_CLK_SLEEP_DISABLE
__HAL_RCC_FLITF_CLK_SLEEP_DISABLE
__HAL_RCC_DMA1_CLK_SLEEP_DISABLE
__HAL_RCC_TIM2_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_TIM3_CLK_SLEEP_ENABLE
__HAL_RCC_TIM4_CLK_SLEEP_ENABLE
__HAL_RCC_TIM6_CLK_SLEEP_ENABLE
__HAL_RCC_TIM7_CLK_SLEEP_ENABLE
__HAL_RCC_WWDG_CLK_SLEEP_ENABLE
__HAL_RCC_SPI2_CLK_SLEEP_ENABLE
__HAL_RCC_USART2_CLK_SLEEP_ENABLE
__HAL_RCC_USART3_CLK_SLEEP_ENABLE
__HAL_RCC_I2C1_CLK_SLEEP_ENABLE
__HAL_RCC_I2C2_CLK_SLEEP_ENABLE
__HAL_RCC_USB_CLK_SLEEP_ENABLE
__HAL_RCC_PWR_CLK_SLEEP_ENABLE
__HAL_RCC_DAC_CLK_SLEEP_ENABLE
__HAL_RCC_COMP_CLK_SLEEP_ENABLE
__HAL_RCC_TIM2_CLK_SLEEP_DISABLE
__HAL_RCC_TIM3_CLK_SLEEP_DISABLE
__HAL_RCC_TIM4_CLK_SLEEP_DISABLE
__HAL_RCC_TIM6_CLK_SLEEP_DISABLE
__HAL_RCC_TIM7_CLK_SLEEP_DISABLE
__HAL_RCC_WWDG_CLK_SLEEP_DISABLE
__HAL_RCC_SPI2_CLK_SLEEP_DISABLE
__HAL_RCC_USART2_CLK_SLEEP_DISABLE
__HAL_RCC_USART3_CLK_SLEEP_DISABLE
__HAL_RCC_I2C1_CLK_SLEEP_DISABLE
__HAL_RCC_I2C2_CLK_SLEEP_DISABLE
__HAL_RCC_USB_CLK_SLEEP_DISABLE
__HAL_RCC_PWR_CLK_SLEEP_DISABLE

__HAL_RCC_DAC_CLK_SLEEP_DISABLE
 __HAL_RCC_COMP_CLK_SLEEP_DISABLE
 __HAL_RCC_SYSCFG_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_TIM9_CLK_SLEEP_ENABLE
 __HAL_RCC_TIM10_CLK_SLEEP_ENABLE
 __HAL_RCC_TIM11_CLK_SLEEP_ENABLE
 __HAL_RCC_ADC1_CLK_SLEEP_ENABLE
 __HAL_RCC_SPI1_CLK_SLEEP_ENABLE
 __HAL_RCC_USART1_CLK_SLEEP_ENABLE
 __HAL_RCC_SYSCFG_CLK_SLEEP_DISABLE
 __HAL_RCC_TIM9_CLK_SLEEP_DISABLE
 __HAL_RCC_TIM10_CLK_SLEEP_DISABLE
 __HAL_RCC_TIM11_CLK_SLEEP_DISABLE
 __HAL_RCC_ADC1_CLK_SLEEP_DISABLE
 __HAL_RCC_SPI1_CLK_SLEEP_DISABLE
 __HAL_RCC_USART1_CLK_SLEEP_DISABLE

PLL Clock Source

RCC_PLLSOURCE_HSI HSI clock selected as PLL entry clock source

RCC_PLLSOURCE_HSE HSE clock selected as PLL entry clock source

PLL Config

RCC_PLL_NONE PLL is not configured

RCC_PLL_OFF PLL deactivation

RCC_PLL_ON PLL activation

PLL Configuration

__HAL_RCC_PLL_ENABLE

Notes:

- After enabling the main PLL, the application software should wait on PLLRDY flag to be set indicating that PLL clock is stable and can be used as system clock source. The main PLL is disabled by hardware when entering STOP and STANDBY modes.

__HAL_RCC_PLL_DISABLE

Notes:

- The main PLL can not be disabled if it is

used as system clock source

`__HAL_RCC_PLL_CONFIG`

Description:

- Macro to configure the main PLL clock source, multiplication and division factors.

Parameters:

- `__RCC_PLLSOURCE__`: specifies the PLL entry clock source. This parameter can be one of the following values:
 - `RCC_PLLSOURCE_HSI`: HSI oscillator clock selected as PLL clock entry
 - `RCC_PLLSOURCE_HSE`: HSE oscillator clock selected as PLL clock entry
- `__PLLMUL__`: specifies the multiplication factor for PLL VCO output clock This parameter can be one of the following values:
 - `RCC_PLL_MUL3`: PLLVCO = PLL clock entry x 3
 - `RCC_PLL_MUL4`: PLLVCO = PLL clock entry x 4
 - `RCC_PLL_MUL6`: PLLVCO = PLL clock entry x 6
 - `RCC_PLL_MUL8`: PLLVCO = PLL clock entry x 8
 - `RCC_PLL_MUL12`: PLLVCO = PLL clock entry x 12
 - `RCC_PLL_MUL16`: PLLVCO = PLL clock entry x 16
 - `RCC_PLL_MUL24`: PLLVCO = PLL clock entry x 24
 - `RCC_PLL_MUL32`: PLLVCO = PLL clock entry x 32
 - `RCC_PLL_MUL48`: PLLVCO = PLL clock entry x 48
- `__PLLDIV__`: specifies the division factor for PLL VCO input clock This parameter can be one of the following values:
 - `RCC_PLL_DIV2`: PLL clock output = PLLVCO / 2
 - `RCC_PLL_DIV3`: PLL clock output = PLLVCO / 3
 - `RCC_PLL_DIV4`: PLL clock output = PLLVCO / 4

Notes:

- This function must be used only when the main PLL is disabled.
- The PLL VCO clock frequency must not exceed 96 MHz when the product is in Range 1, 48 MHz when the product is in

Range 2 and 24 MHz when the product is in Range 3.

`__HAL_RCC_GET_PLL_OSCSOURCE`

Description:

- Get oscillator clock selected as PLL input clock.

Return value:

- The clock source used for PLL entry. The returned value can be one of the following:
 - `RCC_PLLSOURCE_HSI`: HSI oscillator clock selected as PLL input clock
 - `RCC_PLLSOURCE_HSE`: HSE oscillator clock selected as PLL input clock

PLL Division Factor

`RCC_PLL_DIV2`

`RCC_PLL_DIV3`

`RCC_PLL_DIV4`

PLL Multiplication Factor

`RCC_PLL_MUL3`

`RCC_PLL_MUL4`

`RCC_PLL_MUL6`

`RCC_PLL_MUL8`

`RCC_PLL_MUL12`

`RCC_PLL_MUL16`

`RCC_PLL_MUL24`

`RCC_PLL_MUL32`

`RCC_PLL_MUL48`

RCC Private Constants

`RCC_CR_BYTE2_ADDRESS`

`RCC_CIR_BYTE1_ADDRESS`

`RCC_CIR_BYTE2_ADDRESS`

`CR_REG_INDEX`

`CSR_REG_INDEX`

`RCC_FLAG_MASK`

RCC Private Macros

`MCO1_CLK_ENABLE`

`MCO1_GPIO_PORT`

`MCO1_PIN`

IS_RCC_PLLSOURCE
IS_RCC_OSCILLATORTYPE
IS_RCC_HSE
IS_RCC_LSE
IS_RCC_HSI
IS_RCC_CALIBRATION_VALUE
IS_RCC_MSICALIBRATION_VALUE
IS_RCC_MSI_CLOCK_RANGE
IS_RCC_LSI
IS_RCC_MSI
IS_RCC_PLL
IS_RCC_PLL_DIV
IS_RCC_PLL_MUL
IS_RCC_CLOCKTYPE
IS_RCC_SYSCLKSOURCE
IS_RCC_HCLK
IS_RCC_PCLK
IS_RCC_MCO
IS_RCC_MCODIV
IS_RCC_MCO1SOURCE
IS_RCC_RTCCLKSOURCE

Register offsets

RCC_OFFSET
RCC_CR_OFFSET
RCC_CFGR_OFFSET
RCC_CIR_OFFSET
RCC_CSR_OFFSET

RCC RTC Clock Configuration

__HAL_RCC_RTC_CLKPRESCALER

Description:

- Macro to configures the RTC clock (RTCCLK).

Parameters:

- __RTC_CLKSOURCE__: specifies the RTC clock source. This parameter can be one of the following values:
 - RCC_RTCCLKSOURCE_NO_CLK: No clock selected as RTC clock
 - RCC_RTCCLKSOURCE_LSE: LSE selected as RTC clock

- RCC_RTCCLKSOURCE_LSI: LSI selected as RTC clock
- RCC_RTCCLKSOURCE_HSE_DIV2: HSE divided by 2 selected as RTC clock
- RCC_RTCCLKSOURCE_HSE_DIV4: HSE divided by 4 selected as RTC clock
- RCC_RTCCLKSOURCE_HSE_DIV8: HSE divided by 8 selected as RTC clock
- RCC_RTCCLKSOURCE_HSE_DIV16: HSE divided by 16 selected as RTC clock

Notes:

- As the RTC clock configuration bits are in the Backup domain and write access is denied to this domain after reset, you have to enable write access using the Power Backup Access macro before to configure the RTC clock source (to be done once after reset). Once the RTC clock is configured it can't be changed unless the Backup domain is reset using `__HAL_RCC_BACKUPRESET_FORCE()` macro, or by a Power On Reset (POR). RTC prescaler cannot be modified if HSE is enabled (`HSEON = 1`).
- If the LSE or LSI is used as RTC clock source, the RTC continues to work in STOP and STANDBY modes, and can be used as wakeup source. However, when the HSE clock is used as RTC clock source, the RTC cannot be used in STOP and STANDBY modes. The maximum input clock frequency for RTC is 1MHz (when using HSE as RTC clock source).

`__HAL_RCC_RTC_CONFIG``__HAL_RCC_GET_RTC_SOURCE`**Description:**

- macros to get the RTC clock source.

Return value:

- The: clock source can be one of the following values:
 - RCC_RTCCLKSOURCE_NO_CLK: No clock selected as RTC clock
 - RCC_RTCCLKSOURCE_LSE: LSE selected as RTC clock
 - RCC_RTCCLKSOURCE_LSI: LSI selected as RTC clock
 - RCC_RTCCLKSOURCE_HSE_DIV2: HSE divided by 2 selected as RTC

- clock
- RCC_RTCCLKSOURCE_HSE_DIV4: HSE divided by 4 selected as RTC clock
- RCC_RTCCLKSOURCE_HSE_DIV8: HSE divided by 8 selected as RTC clock
- RCC_RTCCLKSOURCE_HSE_DIV16: HSE divided by 16 selected as RTC clock

`__HAL_RCC_RTC_ENABLE`

Notes:

- These macros must be used only after the RTC clock source was selected.

`__HAL_RCC_RTC_DISABLE`

Notes:

- These macros must be used only after the RTC clock source was selected.

`__HAL_RCC_BACKUPRESET_FORCE`

Notes:

- This function resets the RTC peripheral (including the backup registers) and the RTC clock source selection in RCC_CSR register. The BKPSRAM is not affected by this reset.

`__HAL_RCC_BACKUPRESET_RELEASE`

RTC LCD Clock Source

<code>RCC_RTCCLKSOURCE_NO_CLK</code>	No clock
<code>RCC_RTCCLKSOURCE_LSE</code>	LSE oscillator clock used as RTC clock
<code>RCC_RTCCLKSOURCE_LSI</code>	LSI oscillator clock used as RTC clock
<code>RCC_RTCCLKSOURCE_HSE_DIV2</code>	HSE oscillator clock divided by 2 used as RTC clock
<code>RCC_RTCCLKSOURCE_HSE_DIV4</code>	HSE oscillator clock divided by 4 used as RTC clock
<code>RCC_RTCCLKSOURCE_HSE_DIV8</code>	HSE oscillator clock divided by 8 used as RTC clock
<code>RCC_RTCCLKSOURCE_HSE_DIV16</code>	HSE oscillator clock divided by 16 used as RTC clock

System Clock Source

<code>RCC_SYSCLKSOURCE_MSI</code>	MSI selected as system clock
<code>RCC_SYSCLKSOURCE_HSI</code>	HSI selected as system clock
<code>RCC_SYSCLKSOURCE_HSE</code>	HSE selected as system clock
<code>RCC_SYSCLKSOURCE_PLLCLK</code>	PLL selected as system clock

System Clock Source Status

`RCC_SYSCLKSOURCE_STATUS_MSI`

RCC_SYSCLKSOURCE_STATUS_HSI
RCC_SYSCLKSOURCE_STATUS_HSE
RCC_SYSCLKSOURCE_STATUS_PLLCLK

System Clock Type

RCC_CLOCKTYPE_SYSCLK SYSCLK to configure
RCC_CLOCKTYPE_HCLK HCLK to configure
RCC_CLOCKTYPE_PCLK1 PCLK1 to configure
RCC_CLOCKTYPE_PCLK2 PCLK2 to configure

RCC Timeout

RCC_DBP_TIMEOUT_VALUE
RCC_LSE_TIMEOUT_VALUE
CLOCKSWITCH_TIMEOUT_VALUE
HSE_TIMEOUT_VALUE
MSI_TIMEOUT_VALUE
HSI_TIMEOUT_VALUE
LSI_TIMEOUT_VALUE
PLL_TIMEOUT_VALUE

34 HAL RCC Extension Driver

34.1 HAL RCC Extension Driver

34.2 RCCEX Firmware driver registers structures

34.2.1 RCC_PeriphCLKInitTypeDef

Data Fields

- *uint32_t PeriphClockSelection*
- *uint32_t RTCClockSelection*
- *uint32_t LCDClockSelection*

Field Documentation

- *uint32_t RCC_PeriphCLKInitTypeDef::PeriphClockSelection*
The Extended Clock to be configured. This parameter can be a value of [RCCEX_Periph_Clock_Selection](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::RTCClockSelection*
specifies the RTC clock source. This parameter can be a value of [RCC_RTC_LCD_Clock_Source](#)
- *uint32_t RCC_PeriphCLKInitTypeDef::LCDClockSelection*
specifies the LCD clock source. This parameter can be a value of [RCC_RTC_LCD_Clock_Source](#)

34.3 RCCEX Firmware driver API description

34.3.1 Extended Peripheral Control functions

This subsection provides a set of functions allowing to control the RCC Clocks frequencies.



Important note: Care must be taken when HAL_RCCEX_PeriphCLKConfig() is used to select the RTC clock source; in this case the Backup domain will be reset in order to modify the RTC Clock source, as consequence RTC registers (including the backup registers) and RCC_BDCR register are set to their reset values.

This section contains the following APIs:

- [HAL_RCCEX_PeriphCLKConfig\(\)](#)
- [HAL_RCCEX_GetPeriphCLKConfig\(\)](#)
- [HAL_RCCEX_GetPeriphCLKFreq\(\)](#)
- [HAL_RCCEX_EnableLSECSS\(\)](#)
- [HAL_RCCEX_DisableLSECSS\(\)](#)

34.3.2 HAL_RCCEX_PeriphCLKConfig

Function Name	HAL_StatusTypeDef HAL_RCCEX_PeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)
Function Description	Initializes the RCC extended peripherals clocks according to the specified parameters in the RCC_PeriphCLKInitTypeDef.
Parameters	<ul style="list-style-type: none"> • PeriphClkInit: pointer to an RCC_PeriphCLKInitTypeDef structure that contains the configuration information for the Extended Peripherals clocks(RTC/LCD clock).
Return values	<ul style="list-style-type: none"> • HAL status
34.3.3 HAL_RCCEX_GetPeriphCLKConfig	
Function Name	void HAL_RCCEX_GetPeriphCLKConfig (RCC_PeriphCLKInitTypeDef * PeriphClkInit)
Function Description	Get the PeriphClkInit according to the internal RCC configuration registers.
Parameters	<ul style="list-style-type: none"> • PeriphClkInit: pointer to an RCC_PeriphCLKInitTypeDef structure that returns the configuration information for the Extended Peripherals clocks(RTC/LCD clocks).
Return values	<ul style="list-style-type: none"> • None
34.3.4 HAL_RCCEX_GetPeriphCLKFreq	
Function Name	uint32_t HAL_RCCEX_GetPeriphCLKFreq (uint32_t PeriphClk)
Function Description	Returns the peripheral clock frequency.
Parameters	<ul style="list-style-type: none"> • PeriphClk: Peripheral clock identifier This parameter can be one of the following values: RCC_PERIPHCLK_RTC: RTC peripheral clockRCC_PERIPHCLK_LCD: LCD peripheral clock (depends on devices)
Return values	<ul style="list-style-type: none"> • Frequency in Hz (0: means that no available frequency for the peripheral)
Notes	<ul style="list-style-type: none"> • Returns 0 if peripheral clock is unknown
34.3.5 HAL_RCCEX_EnableLSECSS	
Function Name	void HAL_RCCEX_EnableLSECSS (void)
Function Description	Enables the LSE Clock Security System.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • If a failure is detected on the external 32 kHz oscillator, the LSE clock is no longer supplied to the RTC but no hardware action is made to the registers. In Standby mode a wakeup is generated. In other modes an interrupt can be sent to wakeup the software (see Section 5.3.4: Clock interrupt register (RCC_CIR) on page 104). The software MUST then disable the LSECSSON bit, stop the defective 32 kHz oscillator (disabling LSEON), and can change the RTC clock source (no clock or LSI or HSE, with RTCSEL), or take any required action to secure the application. • LSE CSS available only for high density and medium+

34.3.6 HAL_RCCEX_DisableLSECSS

Function Name	void HAL_RCCEX_DisableLSECSS (void)
Function Description	Disables the LSE Clock Security System.
Return values	<ul style="list-style-type: none"> • None
Notes	<ul style="list-style-type: none"> • Once enabled this bit cannot be disabled, except after an LSE failure detection (LSECSSD=1). In that case the software MUST disable the LSECSSON bit. Reset by power on reset and RTC software reset (RTCRST bit). • LSE CSS available only for high density and medium+ devices

34.4 RCCEX Firmware driver defines

34.4.1 RCCEX

RCCEX Force Release Peripheral Reset

```

__HAL_RCC_GPIOE_FORCE_RESET
__HAL_RCC_GPIOE_RELEASE_RESET
__HAL_RCC_GPIOF_FORCE_RESET
__HAL_RCC_GPIOG_FORCE_RESET
__HAL_RCC_GPIOF_RELEASE_RESET
__HAL_RCC_GPIOG_RELEASE_RESET
__HAL_RCC_DMA2_FORCE_RESET
__HAL_RCC_DMA2_RELEASE_RESET
__HAL_RCC_Cryp_FORCE_RESET
__HAL_RCC_Cryp_RELEASE_RESET
__HAL_RCC_FSMC_FORCE_RESET
__HAL_RCC_FSMC_RELEASE_RESET
__HAL_RCC_LCD_FORCE_RESET
__HAL_RCC_LCD_RELEASE_RESET
__HAL_RCC_TIM5_FORCE_RESET
__HAL_RCC_TIM5_RELEASE_RESET
__HAL_RCC_SPI3_FORCE_RESET
__HAL_RCC_SPI3_RELEASE_RESET
__HAL_RCC_UART4_FORCE_RESET
__HAL_RCC_UART5_FORCE_RESET
__HAL_RCC_UART4_RELEASE_RESET
__HAL_RCC_UART5_RELEASE_RESET

```

__HAL_RCC_OPAMP_FORCE_RESET
 __HAL_RCC_OPAMP_RELEASE_RESET
 __HAL_RCC_SDIO_FORCE_RESET
 __HAL_RCC_SDIO_RELEASE_RESET

LCd Configuration

__HAL_RCC_LCD_CONFIG

Description:

- Macro to configures LCD clock (LCDCLK).

Parameters:

- __LCD_CLKSOURCE__: specifies the LCD clock source. This parameter can be one of the following values:
 - RCC_RTCCLKSOURCE_LSE: LSE selected as RTC clock
 - RCC_RTCCLKSOURCE_LSI: LSI selected as RTC clock
 - RCC_RTCCLKSOURCE_HSE_DIV2: HSE divided by 2 selected as RTC clock
 - RCC_RTCCLKSOURCE_HSE_DIV4: HSE divided by 4 selected as RTC clock
 - RCC_RTCCLKSOURCE_HSE_DIV8: HSE divided by 8 selected as RTC clock
 - RCC_RTCCLKSOURCE_HSE_DIV16: HSE divided by 16 selected as RTC clock

Notes:

- LCD and RTC use the same configuration LCD can however be used in the Stop low power mode if the LSE or LSI is used as the LCD clock source.

__HAL_RCC_GET_LCD_SOURCE

RCCEX_Peripheral_Clock_Enable_Disable

__HAL_RCC_GPIOE_CLK_ENABLE
 __HAL_RCC_GPIOE_CLK_DISABLE
 __HAL_RCC_GPIOF_CLK_ENABLE
 __HAL_RCC_GPIOG_CLK_ENABLE
 __HAL_RCC_GPIOF_CLK_DISABLE
 __HAL_RCC_GPIOG_CLK_DISABLE
 __HAL_RCC_DMA2_CLK_ENABLE
 __HAL_RCC_DMA2_CLK_DISABLE
 __HAL_RCC_CRYP_CLK_ENABLE
 __HAL_RCC_CRYP_CLK_DISABLE
 __HAL_RCC_FSMC_CLK_ENABLE

__HAL_RCC_FSMC_CLK_DISABLE
__HAL_RCC_LCD_CLK_ENABLE
__HAL_RCC_LCD_CLK_DISABLE
__HAL_RCC_TIM5_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_TIM5_CLK_DISABLE
__HAL_RCC_SPI3_CLK_ENABLE
__HAL_RCC_SPI3_CLK_DISABLE
__HAL_RCC_UART4_CLK_ENABLE
__HAL_RCC_UART5_CLK_ENABLE
__HAL_RCC_UART4_CLK_DISABLE
__HAL_RCC_UART5_CLK_DISABLE
__HAL_RCC_OPAMP_CLK_ENABLE
__HAL_RCC_OPAMP_CLK_DISABLE
__HAL_RCC_SDIO_CLK_ENABLE

Notes:

- After reset, the peripheral clock (used for registers read/write access) is disabled and the application software has to enable this clock before using it.

__HAL_RCC_SDIO_CLK_DISABLE

Peripheral Clock Enable Disable Status

__HAL_RCC_GPIOE_IS_CLK_ENABLED
__HAL_RCC_GPIOE_IS_CLK_DISABLED
__HAL_RCC_GPIOF_IS_CLK_ENABLED
__HAL_RCC_GPIOG_IS_CLK_ENABLED
__HAL_RCC_GPIOF_IS_CLK_DISABLED
__HAL_RCC_GPIOG_IS_CLK_DISABLED
__HAL_RCC_DMA2_IS_CLK_ENABLED
__HAL_RCC_DMA2_IS_CLK_DISABLED
__HAL_RCC_CRYP_IS_CLK_ENABLED
__HAL_RCC_CRYP_IS_CLK_DISABLED
__HAL_RCC_FSMC_IS_CLK_ENABLED
__HAL_RCC_FSMC_IS_CLK_DISABLED
__HAL_RCC_LCD_IS_CLK_ENABLED
__HAL_RCC_LCD_IS_CLK_DISABLED

__HAL_RCC_TIM5_IS_CLK_ENABLED
__HAL_RCC_TIM5_IS_CLK_DISABLED
__HAL_RCC_SPI3_IS_CLK_ENABLED
__HAL_RCC_SPI3_IS_CLK_DISABLED
__HAL_RCC_UART4_IS_CLK_ENABLED
__HAL_RCC_UART5_IS_CLK_ENABLED
__HAL_RCC_UART4_IS_CLK_DISABLED
__HAL_RCC_UART5_IS_CLK_DISABLED
__HAL_RCC_OPAMP_IS_CLK_ENABLED
__HAL_RCC_OPAMP_IS_CLK_DISABLED
__HAL_RCC_SDIO_IS_CLK_ENABLED
__HAL_RCC_SDIO_IS_CLK_DISABLED

RCCEx Peripheral Clock Sleep Enable Disable

__HAL_RCC_GPIOE_CLK_SLEEP_ENABLE
__HAL_RCC_GPIOE_CLK_SLEEP_DISABLE
__HAL_RCC_GPIOF_CLK_SLEEP_ENABLE
__HAL_RCC_GPIOG_CLK_SLEEP_ENABLE
__HAL_RCC_GPIOF_CLK_SLEEP_DISABLE
__HAL_RCC_GPIOG_CLK_SLEEP_DISABLE
__HAL_RCC_DMA2_CLK_SLEEP_ENABLE
__HAL_RCC_DMA2_CLK_SLEEP_DISABLE
__HAL_RCC_Cryp_CLK_SLEEP_ENABLE
__HAL_RCC_Cryp_CLK_SLEEP_DISABLE
__HAL_RCC_FSMC_CLK_SLEEP_ENABLE
__HAL_RCC_FSMC_CLK_SLEEP_DISABLE
__HAL_RCC_LCD_CLK_SLEEP_ENABLE
__HAL_RCC_LCD_CLK_SLEEP_DISABLE
__HAL_RCC_TIM5_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_TIM5_CLK_SLEEP_DISABLE
__HAL_RCC_SPI3_CLK_SLEEP_ENABLE
__HAL_RCC_SPI3_CLK_SLEEP_DISABLE

__HAL_RCC_UART4_CLK_SLEEP_ENABLE
__HAL_RCC_UART5_CLK_SLEEP_ENABLE
__HAL_RCC_UART4_CLK_SLEEP_DISABLE
__HAL_RCC_UART5_CLK_SLEEP_DISABLE
__HAL_RCC_OPAMP_CLK_SLEEP_ENABLE
__HAL_RCC_OPAMP_CLK_SLEEP_DISABLE
__HAL_RCC_SDIO_CLK_SLEEP_ENABLE

Notes:

- Peripheral clock gating in SLEEP mode can be used to further reduce power consumption. After wakeup from SLEEP mode, the peripheral clock is enabled again. By default, all peripheral clocks are enabled during SLEEP mode.

__HAL_RCC_SDIO_CLK_SLEEP_DISABLE

Peripheral Clock Sleep Enable Disable Status

__HAL_RCC_GPIOE_IS_CLK_SLEEP_ENABLED
__HAL_RCC_GPIOE_IS_CLK_SLEEP_DISABLED
__HAL_RCC_GPIOF_IS_CLK_SLEEP_ENABLED
__HAL_RCC_GPIOG_IS_CLK_SLEEP_ENABLED
__HAL_RCC_GPIOF_IS_CLK_SLEEP_DISABLED
__HAL_RCC_GPIOG_IS_CLK_SLEEP_DISABLED
__HAL_RCC_DMA2_IS_CLK_SLEEP_ENABLED
__HAL_RCC_DMA2_IS_CLK_SLEEP_DISABLED
__HAL_RCC_Cryp_IS_CLK_SLEEP_ENABLED
__HAL_RCC_Cryp_IS_CLK_SLEEP_DISABLED
__HAL_RCC_FSMC_IS_CLK_SLEEP_ENABLED
__HAL_RCC_FSMC_IS_CLK_SLEEP_DISABLED
__HAL_RCC_LCD_IS_CLK_SLEEP_ENABLED
__HAL_RCC_LCD_IS_CLK_SLEEP_DISABLED
__HAL_RCC_TIM5_IS_CLK_SLEEP_ENABLED
__HAL_RCC_TIM5_IS_CLK_SLEEP_DISABLED
__HAL_RCC_SPI3_IS_CLK_SLEEP_ENABLED
__HAL_RCC_SPI3_IS_CLK_SLEEP_DISABLED
__HAL_RCC_UART4_IS_CLK_SLEEP_ENABLED
__HAL_RCC_UART5_IS_CLK_SLEEP_ENABLED
__HAL_RCC_UART4_IS_CLK_SLEEP_DISABLED
__HAL_RCC_UART5_IS_CLK_SLEEP_DISABLED

__HAL_RCC_OPAMP_IS_CLK_SLEEP_ENABLED

__HAL_RCC_OPAMP_IS_CLK_SLEEP_DISABLED

__HAL_RCC_SDIO_IS_CLK_SLEEP_ENABLED

__HAL_RCC_SDIO_IS_CLK_SLEEP_DISABLED

RCCEx Periph Clock Selection

RCC_PERIPHCLK_RTC

RCC_PERIPHCLK_LCD

RCCEx Private Constants

LSI_VALUE

LSECSSON_BITNUMBER

CSR_LSECSSON_BB

RCCEx Private Macros

IS_RCC_PERIPHCLK

35 HAL RTC Generic Driver

35.1 HAL RTC Generic Driver

35.2 RTC Firmware driver registers structures

35.2.1 RTC_InitTypeDef

Data Fields

- *uint32_t HourFormat*
- *uint32_t AsynchPrediv*
- *uint32_t SynchPrediv*
- *uint32_t OutPut*
- *uint32_t OutPutPolarity*
- *uint32_t OutPutType*

Field Documentation

- *uint32_t RTC_InitTypeDef::HourFormat*
Specifies the RTC Hour Format. This parameter can be a value of [RTC_Hour_Formats](#)
- *uint32_t RTC_InitTypeDef::AsynchPrediv*
Specifies the RTC Asynchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7F
- *uint32_t RTC_InitTypeDef::SynchPrediv*
Specifies the RTC Synchronous Predivider value. This parameter must be a number between Min_Data = 0x00 and Max_Data = 0x7FFF
- *uint32_t RTC_InitTypeDef::OutPut*
Specifies which signal will be routed to the RTC output. This parameter can be a value of [RTCEx_Output_selection_Definitions](#)
- *uint32_t RTC_InitTypeDef::OutPutPolarity*
Specifies the polarity of the output signal. This parameter can be a value of [RTC_Output_Polarity_Definitions](#)
- *uint32_t RTC_InitTypeDef::OutPutType*
Specifies the RTC Output Pin mode. This parameter can be a value of [RTC_Output_Type_ALARM_OUT](#)

35.2.2 RTC_DateTypeDef

Data Fields

- *uint8_t WeekDay*
- *uint8_t Month*
- *uint8_t Date*
- *uint8_t Year*

Field Documentation

- ***uint8_t RTC_DateTypeDef::WeekDay***
Specifies the RTC Date WeekDay. This parameter can be a value of [RTC_WeekDay_Definitions](#)
- ***uint8_t RTC_DateTypeDef::Month***
Specifies the RTC Date Month (in BCD format). This parameter can be a value of [RTC_Month_Date_Definitions](#)
- ***uint8_t RTC_DateTypeDef::Date***
Specifies the RTC Date. This parameter must be a number between Min_Data = 1 and Max_Data = 31
- ***uint8_t RTC_DateTypeDef::Year***
Specifies the RTC Date Year. This parameter must be a number between Min_Data = 0 and Max_Data = 99

35.2.3 RTC_HandleTypeDef**Data Fields**

- ***RTC_TypeDef * Instance***
- ***RTC_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_RTCStateTypeDef State***

Field Documentation

- ***RTC_TypeDef* RTC_HandleTypeDef::Instance***
Register base address
- ***RTC_InitTypeDef RTC_HandleTypeDef::Init***
RTC required parameters
- ***HAL_LockTypeDef RTC_HandleTypeDef::Lock***
RTC locking object
- ***__IO HAL_RTCStateTypeDef RTC_HandleTypeDef::State***
Time communication state

35.3 RTC Firmware driver API description**35.3.1 Backup Domain Operating Condition**

The real-time clock (RTC) and the RTC backup registers can be powered from the VBAT voltage when the main VDD supply is powered off. To retain the content of the RTC backup registers and supply the RTC when VDD is turned off, VBAT pin can be connected to an optional standby voltage supplied by a battery or by another source.

To allow the RTC operating even when the main digital supply (VDD) is turned off, the VBAT pin powers the following blocks:

1. The RTC
2. The LSE oscillator
3. PC13 to PC15 I/Os (when available)

When the backup domain is supplied by VDD (analog switch connected to VDD), the following pins are available:

1. PC14 and PC15 can be used as either GPIO or LSE pins
2. PC13 can be used as a GPIO or as the RTC_AF1 pin

When the backup domain is supplied by VBAT (analog switch connected to VBAT because VDD is not present), the following pins are available:

1. PC14 and PC15 can be used as LSE pins only
2. PC13 can be used as the RTC_AF1 pin

35.3.2 Backup Domain Reset

The backup domain reset sets all RTC registers and the RCC_BDCR register to their reset values.

A backup domain reset is generated when one of the following events occurs:

1. Software reset, triggered by setting the BDRST bit in the RCC Backup domain control register (RCC_BDCR).
2. VDD or VBAT power on, if both supplies have previously been powered off.

35.3.3 Backup Domain Access

After reset, the backup domain (RTC registers, RTC backup data registers and backup SRAM) is protected against possible unwanted write accesses.

To enable access to the RTC Domain and RTC registers, proceed as follows:

- Enable the Power Controller (PWR) APB1 interface clock using the `__HAL_RCC_PWR_CLK_ENABLE()` function.
- Enable access to RTC domain using the `HAL_PWR_EnableBkUpAccess()` function.
- Select the RTC clock source using the `__HAL_RCC_RTC_CONFIG()` function.
- Enable RTC Clock using the `__HAL_RCC_RTC_ENABLE()` function.

35.3.4 How to use this driver

- Enable the RTC domain access (see description in the section above).
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the `HAL_RTC_Init()` function.

Time and Date configuration

- To configure the RTC Calendar (Time and Date) use the `HAL_RTC_SetTime()` and `HAL_RTC_SetDate()` functions.
- To read the RTC Calendar, use the `HAL_RTC_GetTime()` and `HAL_RTC_GetDate()` functions.

Alarm configuration

- To configure the RTC Alarm use the `HAL_RTC_SetAlarm()` function. You can also configure the RTC Alarm with interrupt mode using the `HAL_RTC_SetAlarm_IT()` function.
- To read the RTC Alarm, use the `HAL_RTC_GetAlarm()` function.

35.3.5 RTC and low power modes

The MCU can be woken up from a low power mode by an RTC alternate function.

The RTC alternate functions are the RTC alarms (Alarm A and Alarm B), RTC wakeup, RTC tamper event detection and RTC time stamp event detection. These RTC alternate functions can wake up the system from the Stop and Standby low power modes.

The system can also wake up from low power modes without depending on an external interrupt (Auto-wakeup mode), by using the RTC alarm or the RTC wakeup events.

The RTC provides a programmable time base for waking up from the Stop or Standby mode at regular intervals. Wakeup from STOP and STANDBY modes is possible only when the RTC clock source is LSE or LSI.

35.3.6 Initialization and de-initialization functions

This section provides functions allowing to initialize and configure the RTC Prescaler (Synchronous and Asynchronous), RTC Hour format, disable RTC registers Write protection, enter and exit the RTC initialization mode, RTC registers synchronization check and reference clock detection enable.

1. The RTC Prescaler is programmed to generate the RTC 1Hz time base. It is split into 2 programmable prescalers to minimize power consumption.
 - A 7-bit asynchronous prescaler and a 13-bit synchronous prescaler.
 - When both prescalers are used, it is recommended to configure the asynchronous prescaler to a high value to minimize power consumption.
2. All RTC registers are Write protected. Writing to the RTC registers is enabled by writing a key into the Write Protection register, RTC_WPR.
3. To configure the RTC Calendar, user application should enter initialization mode. In this mode, the calendar counter is stopped and its value can be updated. When the initialization sequence is complete, the calendar restarts counting after 4 RTCCLK cycles.
4. To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers. The HAL_RTC_WaitForSynchro() function implements the above software sequence (RSF clear and RSF check).

This section contains the following APIs:

- [*HAL_RTC_Init\(\)*](#)
- [*HAL_RTC_DeInit\(\)*](#)
- [*HAL_RTC_MspltInit\(\)*](#)
- [*HAL_RTC_MspDeInit\(\)*](#)
- [*HAL_RTC_SetTime\(\)*](#)
- [*HAL_RTC_GetTime\(\)*](#)
- [*HAL_RTC_SetDate\(\)*](#)
- [*HAL_RTC_GetDate\(\)*](#)

35.3.7 RTC Time and Date functions

This section provides functions allowing to configure Time and Date features

This section contains the following APIs:

- [*HAL_RTC_SetTime\(\)*](#)
- [*HAL_RTC_SetDate\(\)*](#)
- [*HAL_RTC_GetDate\(\)*](#)

- [HAL_RTC_GetTime\(\)](#)
- [HAL_RTC_SetAlarm\(\)](#)
- [HAL_RTC_SetAlarm_IT\(\)](#)
- [HAL_RTC_DeactivateAlarm\(\)](#)
- [HAL_RTC_GetAlarm\(\)](#)
- [HAL_RTC_AlarmIRQHandler\(\)](#)
- [HAL_RTC_PollForAlarmAEvent\(\)](#)
- [HAL_RTC_AlarmAEventCallback\(\)](#)

35.3.8 RTC Alarm functions

This section provides functions allowing to configure Alarm feature

This section contains the following APIs:

- [HAL_RTC_DeactivateAlarm\(\)](#)
- [HAL_RTC_AlarmIRQHandler\(\)](#)
- [HAL_RTC_AlarmAEventCallback\(\)](#)
- [HAL_RTC_PollForAlarmAEvent\(\)](#)
- [HAL_RTC_SetAlarm\(\)](#)
- [HAL_RTC_SetAlarm_IT\(\)](#)
- [HAL_RTC_GetAlarm\(\)](#)
- [HAL_RTC_WaitForSynchro\(\)](#)

35.3.9 Peripheral State functions

This subsection provides functions allowing to

- Get RTC state

This section contains the following APIs:

- [HAL_RTC_GetState\(\)](#)

35.3.10 Peripheral Control functions

This subsection provides functions allowing to

- Wait for RTC Time and Date Synchronization

This section contains the following APIs:

- [HAL_RTC_WaitForSynchro\(\)](#)

35.3.11 HAL_RTC_Init

Function Name	HAL_StatusTypeDef HAL_RTC_Init (RTC_HandleTypeDef * hrtc)
Function Description	Initializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

35.3.12 HAL_RTC_DeInit

Function Name	HAL_StatusTypeDef HAL_RTC_DeInit (RTC_HandleTypeDef * hrtc)
---------------	---

Function Description	DeInitializes the RTC peripheral.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status • HAL status
Notes	<ul style="list-style-type: none"> • This function doesn't reset the RTC Backup Data registers. • This function does not reset the RTC Backup Data registers.

35.3.13 HAL_RTC_MspInit

Function Name	void HAL_RTC_MspInit (RTC_HandleTypeDef * hrtc)
Function Description	Initializes the RTC MSP.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

35.3.14 HAL_RTC_MspDeInit

Function Name	void HAL_RTC_MspDeInit (RTC_HandleTypeDef * hrtc)
Function Description	DeInitializes the RTC MSP.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

35.3.15 HAL_RTC_SetTime

Function Name	HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
Function Description	Sets RTC current time.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTime: Pointer to Time structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

35.3.16 HAL_RTC_GetTime

Function Name	HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
Function Description	Gets RTC current time.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

	<ul style="list-style-type: none"> • sTime: Pointer to Time structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

35.3.17 HAL_RTC_SetDate

Function Name	HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
Function Description	Sets RTC current date.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sDate: Pointer to date structure • Format: specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

35.3.18 HAL_RTC_GetDate

Function Name	HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
Function Description	Gets RTC current date.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sDate: Pointer to Date structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

35.3.19 HAL_RTC_SetTime

Function Name	HAL_StatusTypeDef HAL_RTC_SetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
---------------	---

Function Description	Sets RTC current time.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTime: Pointer to Time structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

35.3.20 HAL_RTC_SetDate

Function Name	HAL_StatusTypeDef HAL_RTC_SetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
Function Description	Sets RTC current date.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sDate: Pointer to date structure • Format: specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

35.3.21 HAL_RTC_GetDate

Function Name	HAL_StatusTypeDef HAL_RTC_GetDate (RTC_HandleTypeDef * hrtc, RTC_DateTypeDef * sDate, uint32_t Format)
Function Description	Gets RTC current date.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sDate: Pointer to Date structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

35.3.22 HAL_RTC_GetTime

Function Name	HAL_StatusTypeDef HAL_RTC_GetTime (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTime, uint32_t Format)
Function Description	Gets RTC current time.

Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sTime: Pointer to Time structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • You must call HAL_RTC_GetDate() after HAL_RTC_GetTime() to unlock the values in the higher-order calendar shadow registers to ensure consistency between the time and date values. Reading RTC current time locks the values in calendar shadow registers until Current date is read.

35.3.23 HAL_RTC_SetAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_SetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
Function Description	Sets the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sAlarm: Pointer to Alarm structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

35.3.24 HAL_RTC_SetAlarm_IT

Function Name	HAL_StatusTypeDef HAL_RTC_SetAlarm_IT (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)
Function Description	Sets the specified RTC Alarm with Interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sAlarm: Pointer to Alarm structure • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()). • The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

35.3.25 HAL_RTC_DeactivateAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)
Function Description	Deactive the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Alarm: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmARTC_ALARM_B: AlarmB
Return values	<ul style="list-style-type: none"> • HAL status

35.3.26 HAL_RTC_GetAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_GetAlarm (RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)
Function Description	Gets the RTC Alarm value and masks.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • sAlarm: Pointer to Date structure • Alarm: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmARTC_ALARM_B: AlarmB • Format: Specifies the format of the entered parameters. This parameter can be one of the following values: RTC_FORMAT_BIN: Binary data formatRTC_FORMAT_BCD: BCD data format
Return values	<ul style="list-style-type: none"> • HAL status

35.3.27 HAL_RTC_AlarmIRQHandler

Function Name	void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	This function handles Alarm interrupt request.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

35.3.28 HAL_RTC_PollForAlarmAEvent

Function Name	HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles AlarmA Polling request.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

35.3.29 HAL_RTC_AlarmAEventCallback

Function Name	void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Alarm A callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

35.3.30 HAL_RTC_DeactivateAlarm

Function Name	HAL_StatusTypeDef HAL_RTC_DeactivateAlarm (RTC_HandleTypeDef * hrtc, uint32_t Alarm)
Function Description	Deactive the specified RTC Alarm.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Alarm: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmARTC_ALARM_B: AlarmB
Return values	<ul style="list-style-type: none"> • HAL status

35.3.31 HAL_RTC_AlarmIRQHandler

Function Name	void HAL_RTC_AlarmIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	This function handles Alarm interrupt request.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

35.3.32 HAL_RTC_AlarmAEventCallback

Function Name	void HAL_RTC_AlarmAEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Alarm A callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

35.3.33 HAL_RTC_PollForAlarmAEvent

Function Name	HAL_StatusTypeDef HAL_RTC_PollForAlarmAEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles AlarmA Polling request.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Timeout: Timeout duration

Return values

- HAL status

35.3.34 HAL_RTC_SetAlarm

Function Name `HAL_StatusTypeDef HAL_RTC_SetAlarm(RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)`

Function Description Sets the specified RTC Alarm.

Parameters

- **hrtc**: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **sAlarm**: Pointer to Alarm structure
- **Format**: Specifies the format of the entered parameters. This parameter can be one of the following values:
RTC_FORMAT_BIN: Binary data
formatRTC_FORMAT_BCD: BCD data format

Return values

- HAL status

35.3.35 HAL_RTC_SetAlarm_IT

Function Name `HAL_StatusTypeDef HAL_RTC_SetAlarm_IT(RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Format)`

Function Description Sets the specified RTC Alarm with Interrupt.

Parameters

- **hrtc**: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **sAlarm**: Pointer to Alarm structure
- **Format**: Specifies the format of the entered parameters. This parameter can be one of the following values:
RTC_FORMAT_BIN: Binary data
formatRTC_FORMAT_BCD: BCD data format

Return values

- HAL status

Notes

- The Alarm register can only be written when the corresponding Alarm is disabled (Use the HAL_RTC_DeactivateAlarm()).
- The HAL_RTC_SetTime() must be called before enabling the Alarm feature.

35.3.36 HAL_RTC_GetAlarm

Function Name `HAL_StatusTypeDef HAL_RTC_GetAlarm(RTC_HandleTypeDef * hrtc, RTC_AlarmTypeDef * sAlarm, uint32_t Alarm, uint32_t Format)`

Function Description Gets the RTC Alarm value and masks.

Parameters

- **hrtc**: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **sAlarm**: Pointer to Date structure
- **Alarm**: Specifies the Alarm. This parameter can be one of the following values: RTC_ALARM_A: AlarmA
AlarmARTC_ALARM_B: AlarmB
- **Format**: Specifies the format of the entered parameters. This

parameter can be one of the following values:
RTC_FORMAT_BIN: Binary data
formatRTC_FORMAT_BCD: BCD data format

Return values

- HAL status

35.3.37 HAL_RTC_WaitForSynchro

Function Name

HAL_StatusTypeDef HAL_RTC_WaitForSynchro
(RTC_HandleTypeDef * hrtc)

Function Description

Waits until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **hrtc**: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- HAL status

Notes

- The RTC Resynchronization mode is write protected, use the `__HAL_RTC_WRITEPROTECTION_DISABLE()` before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

35.3.38 HAL_RTC_GetState

Function Name

HAL_RTCStateTypeDef HAL_RTC_GetState
(RTC_HandleTypeDef * hrtc)

Function Description

Returns the RTC state.

Parameters

- **hrtc**: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- HAL state

35.3.39 HAL_RTC_WaitForSynchro

Function Name

HAL_StatusTypeDef HAL_RTC_WaitForSynchro
(RTC_HandleTypeDef * hrtc)

Function Description

Waits until the RTC Time and Date registers (RTC_TR and RTC_DR) are synchronized with RTC APB clock.

Parameters

- **hrtc**: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- HAL status

Notes

- The RTC Resynchronization mode is write protected, use the `__HAL_RTC_WRITEPROTECTION_DISABLE()` before calling this function.
- To read the calendar through the shadow registers after Calendar initialization, calendar update or after wakeup from

low power modes the software must first clear the RSF flag. The software must then wait until it is set again before reading the calendar, which means that the calendar registers have been correctly copied into the RTC_TR and RTC_DR shadow registers.

35.4 RTC Firmware driver defines

35.4.1 RTC

AlarmDateWeekDay Definitions

RTC_ALARMDATEWEEKDAYSEL_DATE

RTC_ALARMDATEWEEKDAYSEL_WEEKDAY

IS_RTC_ALARM_DATE_WEEKDAY_SEL

Alarm Mask Definitions

RTC_ALARM_MASK_NONE

RTC_ALARM_MASK_DATEWEEKDAY

RTC_ALARM_MASK_HOURS

RTC_ALARM_MASK_MINUTES

RTC_ALARM_MASK_SECONDS

RTC_ALARM_MASK_ALL

IS_RTC_ALARM_MASK

Alarms Definitions

RTC_ALARM_A

RTC_ALARM_B

IS_RTC_ALARM

Alarm Definitions

IS_RTC_ALARM_DATE_WEEKDAY_DATE

IS_RTC_ALARM_DATE_WEEKDAY_WEEKDAY

Alarm Sub Seconds Masks Definitions

RTC_ALARMSSUBSECONDMASK_ALL	All Alarm SS fields are masked. There is no comparison on sub seconds for Alarm
RTC_ALARMSSUBSECONDMASK_SS14_1	SS[14:1] are don't care in Alarm comparison. Only SS[0] is compared.
RTC_ALARMSSUBSECONDMASK_SS14_2	SS[14:2] are don't care in Alarm comparison. Only SS[1:0] are compared
RTC_ALARMSSUBSECONDMASK_SS14_3	SS[14:3] are don't care in Alarm comparison. Only SS[2:0] are compared
RTC_ALARMSSUBSECONDMASK_SS14_4	SS[14:4] are don't care in Alarm comparison. Only SS[3:0] are compared
RTC_ALARMSSUBSECONDMASK_SS14_5	SS[14:5] are don't care in Alarm

RTC_ALARMSUBSECONDMASK_SS14_6	comparison. Only SS[4:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_7	SS[14:6] are don't care in Alarm comparison. Only SS[5:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_8	SS[14:7] are don't care in Alarm comparison. Only SS[6:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_9	SS[14:8] are don't care in Alarm comparison. Only SS[7:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_10	SS[14:9] are don't care in Alarm comparison. Only SS[8:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_11	SS[14:10] are don't care in Alarm comparison. Only SS[9:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_12	SS[14:11] are don't care in Alarm comparison. Only SS[10:0] are compared
RTC_ALARMSUBSECONDMASK_SS14_13	SS[14:12] are don't care in Alarm comparison. Only SS[11:0] are compared
RTC_ALARMSUBSECONDMASK_SS14	SS[14:13] are don't care in Alarm comparison. Only SS[12:0] are compared
RTC_ALARMSUBSECONDMASK_NONE	SS[14] is don't care in Alarm comparison. Only SS[13:0] are compared
IS_RTC_ALARM_SUB_SECOND_MASK	SS[14:0] are compared and must match to activate alarm.

IS_RTC_ALARM_SUB_SECOND_MASK

Alarm Sub Seconds Value

IS_RTC_ALARM_SUB_SECOND_VALUE

AM PM Definitions

RTC_HOURFORMAT12_AM

RTC_HOURFORMAT12_PM

IS_RTC_HOURFORMAT12

Asynchronous Predivider

IS_RTC_ASYNC_PREDIV

DayLightSaving

RTC_DAYLIGHTSAVING_SUB1H

RTC_DAYLIGHTSAVING_ADD1H

RTC_DAYLIGHTSAVING_NONE

IS_RTC_DAYLIGHT_SAVING

RTC Exported Macros

__HAL_RTC_RESET_HANDLE_STATE

Description:

- Reset RTC handle state.

Parameters:

- __HANDLE__: RTC

handle.

Return value:

- None

Description:

- Disable the write protection for RTC registers.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Enable the write protection for RTC registers.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Enable the RTC ALARMA peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Disable the RTC ALARMA peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Enable the RTC ALARMB peripheral.

`__HAL_RTC_WRITEPROTECTION_DISABLE`

`__HAL_RTC_WRITEPROTECTION_ENABLE`

`__HAL_RTC_ALARMA_ENABLE`

`__HAL_RTC_ALARMA_DISABLE`

`__HAL_RTC_ALARMB_ENABLE`

`__HAL_RTC_ALARMB_DISABLE`

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Disable the RTC ALARMB peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Enable the RTC Alarm interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:
 - `RTC_IT_ALRA`: Alarm A interrupt
 - `RTC_IT_ALRB`: Alarm B interrupt

Return value:

- None

Description:

- Disable the RTC Alarm interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Alarm interrupt sources to be enabled or disabled. This parameter can be any combination of the following values:

`__HAL_RTC_ALARM_ENABLE_IT`

`__HAL_RTC_ALARM_DISABLE_IT`

- RTC_IT_ALRA:
Alarm A interrupt
- RTC_IT_ALRB:
Alarm B interrupt

Return value:

- None

Description:

- Check whether the specified RTC Alarm interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
 - RTC_IT_ALRA:
Alarm A interrupt
 - RTC_IT_ALRB:
Alarm B interrupt

Return value:

- None

Description:

- Get the selected RTC Alarm's flag status.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to check. This parameter can be:
 - RTC_FLAG_ALRAF
 - RTC_FLAG_ALRBF
 - RTC_FLAG_ALRAW
F
 - RTC_FLAG_ALRBW
F

Return value:

- None

Description:

- Clear the RTC Alarm's pending flags.

__HAL_RTC_ALARM_GET_IT

__HAL_RTC_ALARM_GET_FLAG

__HAL_RTC_ALARM_CLEAR_FLAG

__HAL_RTC_ALARM_GET_IT_SOURCE

Parameters:

- __HANDLE__: specifies the RTC handle.
- __FLAG__: specifies the RTC Alarm Flag sources to clear. This parameter can be:
 - RTC_FLAG_ALRAF
 - RTC_FLAG_ALRBF

Return value:

- None

Description:

- Check whether the specified RTC Alarm interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Alarm interrupt sources to check. This parameter can be:
 - RTC_IT_ALRA: Alarm A interrupt
 - RTC_IT_ALRB: Alarm B interrupt

Return value:

- None

Description:

- Enable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

Description:

- Disable interrupt on the RTC Alarm associated Exti line.

Return value:

- None

Description:

- Enable event on the RTC Alarm associated Exti line.

__HAL_RTC_ALARM_EXTI_ENABLE_IT

__HAL_RTC_ALARM_EXTI_DISABLE_IT

__HAL_RTC_ALARM_EXTI_ENABLE_EVENT

__HAL_RTC_ALARM_EXTI_DISABLE_EVENT

Return value:

- None.

Description:

- Disable event on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_ENABLE_FALLING_EDGE

Description:

- Enable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_FALLING_EDGE

Description:

- Disable falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_EDGE

Description:

- Enable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_EDGE

Description:

- Disable rising edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_ENABLE_RISING_FALLING_EDGE

Description:

- Enable rising & falling edge trigger on the RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_DISABLE_RISING_FALLING_EDGE

Description:

- Disable rising & falling edge trigger on the RTC

Alarm associated Exti line.

Return value:

- None.

Description:

- Check whether the RTC Alarm associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

Description:

- Clear the RTC Alarm associated Exti line flag.

Return value:

- None.

Description:

- Generate a Software interrupt on RTC Alarm associated Exti line.

Return value:

- None.

__HAL_RTC_ALARM_EXTI_GET_FLAG

__HAL_RTC_ALARM_EXTI_CLEAR_FLAG

__HAL_RTC_ALARM_EXTI_GENERATE_SWIT

Flags Definitions

RTC_FLAG_RECALPF

RTC_FLAG_TAMP3F

RTC_FLAG_TAMP2F

RTC_FLAG_TAMP1F

RTC_FLAG_TSOVF

RTC_FLAG_TSF

RTC_FLAG_WUTF

RTC_FLAG_ALRBF

RTC_FLAG_ALRAF

RTC_FLAG_INITF

RTC_FLAG_RSF

RTC_FLAG_INITS

RTC_FLAG_SHPF

RTC_FLAG_WUTWF

RTC_FLAG_ALRBWF

RTC_FLAG_ALRAWF

Hour Formats

RTC_HOURFORMAT_24

RTC_HOURFORMAT_12

IS_RTC_HOUR_FORMAT

Input Parameter Format

RTC_FORMAT_BIN

RTC_FORMAT_BCD

IS_RTC_FORMAT

Interrupts Definitions

RTC_IT_TS

RTC_IT_WUT

RTC_IT_ALRB

RTC_IT_ALRA

RTC_IT_TAMP1

RTC_IT_TAMP2

RTC_IT_TAMP3

Masks Definitions

RTC_TR_RESERVED_MASK

RTC_DR_RESERVED_MASK

RTC_INIT_MASK

RTC_RSF_MASK

RTC_FLAGS_MASK

Month Definitions

RTC_MONTH_JANUARY

RTC_MONTH_FEBRUARY

RTC_MONTH_MARCH

RTC_MONTH_APRIL

RTC_MONTH_MAY

RTC_MONTH_JUNE

RTC_MONTH_JULY

RTC_MONTH_AUGUST

RTC_MONTH_SEPTEMBER

RTC_MONTH_OCTOBER

RTC_MONTH_NOVEMBER

RTC_MONTH_DECEMBER

IS_RTC_MONTH

IS_RTC_DATE

Output Polarity

RTC_OUTPUT_POLARITY_HIGH

RTC_OUTPUT_POLARITY_LOW

IS_RTC_OUTPUT_POL

Alarm Output Type

RTC_OUTPUT_TYPE_OPENDRAIN

RTC_OUTPUT_TYPE_PUSHPULL

IS_RTC_OUTPUT_TYPE

RTC Private Constants

RTC_EXTI_LINE_ALARM_EVENT External interrupt line 17 Connected to the RTC
Alarm event

StoreOperation

RTC_STOREOPERATION_RESET

RTC_STOREOPERATION_SET

IS_RTC_STORE_OPERATION

Synchronous Predivider

IS_RTC_SYNCH_PREDIV

Default Timeout Value

RTC_TIMEOUT_VALUE

Time Definitions

IS_RTC_HOUR12

IS_RTC_HOUR24

IS_RTC_MINUTES

IS_RTC_SECONDS

WeekDay Definitions

RTC_WEEKDAY_MONDAY

RTC_WEEKDAY_TUESDAY

RTC_WEEKDAY_WEDNESDAY

RTC_WEEKDAY_THURSDAY

RTC_WEEKDAY_FRIDAY

RTC_WEEKDAY_SATURDAY

RTC_WEEKDAY_SUNDAY

IS_RTC_WEEKDAY

Year Definitions

IS_RTC_YEAR

36 HAL RTC Extension Driver

36.1 HAL RTC Extension Driver

36.2 RTCEX Firmware driver registers structures

36.2.1 RTC_TamperTypeDef

Data Fields

- *uint32_t Tamper*
- *uint32_t Trigger*
- *uint32_t Filter*
- *uint32_t SamplingFrequency*
- *uint32_t PrechargeDuration*
- *uint32_t TamperPullUp*
- *uint32_t TimeStampOnTamperDetection*

Field Documentation

- *uint32_t RTC_TamperTypeDef::Tamper*
Specifies the Tamper Pin. This parameter can be a value of [RTCEX_Tamper_Pins_Definitions](#)
- *uint32_t RTC_TamperTypeDef::Trigger*
Specifies the Tamper Trigger. This parameter can be a value of [RTCEX_Tamper_Trigger_Definitions](#)
- *uint32_t RTC_TamperTypeDef::Filter*
Specifies the RTC Filter Tamper. This parameter can be a value of [RTCEX_Tamper_Filter_Definitions](#)
- *uint32_t RTC_TamperTypeDef::SamplingFrequency*
Specifies the sampling frequency. This parameter can be a value of [RTCEX_Tamper_Sampling_Frequencies_Definitions](#)
- *uint32_t RTC_TamperTypeDef::PrechargeDuration*
Specifies the Precharge Duration . This parameter can be a value of [RTCEX_Tamper_Pin_Precharge_Duration_Definitions](#)
- *uint32_t RTC_TamperTypeDef::TamperPullUp*
Specifies the Tamper PullUp . This parameter can be a value of [RTCEX_Tamper_Pull_Up_Definitions](#)
- *uint32_t RTC_TamperTypeDef::TimeStampOnTamperDetection*
Specifies the TimeStampOnTamperDetection. This parameter can be a value of [RTCEX_Tamper_TimeStampOnTamperDetection_Definitions](#)

36.2.2 RTC_TimeTypeDef

Data Fields

- *uint8_t Hours*
- *uint8_t Minutes*

- ***uint8_t Seconds***
- ***uint32_t SubSeconds***
- ***uint8_t TimeFormat***
- ***uint32_t DayLightSaving***
- ***uint32_t StoreOperation***

Field Documentation

- ***uint8_t RTC_TimeTypeDef::Hours***
Specifies the RTC Time Hour. This parameter must be a number between Min_Data = 0 and Max_Data = 12 if the RTC_HourFormat_12 is selected. This parameter must be a number between Min_Data = 0 and Max_Data = 23 if the RTC_HourFormat_24 is selected.
- ***uint8_t RTC_TimeTypeDef::Minutes***
Specifies the RTC Time Minutes. This parameter must be a number between Min_Data = 0 and Max_Data = 59.
- ***uint8_t RTC_TimeTypeDef::Seconds***
Specifies the RTC Time Seconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59.
- ***uint32_t RTC_TimeTypeDef::SubSeconds***
Specifies the RTC Time SubSeconds. This parameter must be a number between Min_Data = 0 and Max_Data = 59.
- ***uint8_t RTC_TimeTypeDef::TimeFormat***
Specifies the RTC AM/PM Time. This parameter can be a value of [RTC_AM_PM_Definitions](#).
- ***uint32_t RTC_TimeTypeDef::DayLightSaving***
Specifies RTC_DayLightSaveOperation: the value of hour adjustment. This parameter can be a value of [RTC_DayLightSaving_Definitions](#).
- ***uint32_t RTC_TimeTypeDef::StoreOperation***
Specifies RTC_StoreOperation value to be written in the BCK bit in CR register to store the operation. This parameter can be a value of [RTC_StoreOperation_Definitions](#).

36.2.3 RTC_AlarmTypeDef

Data Fields

- ***RTC_TimeTypeDef AlarmTime***
- ***uint32_t AlarmMask***
- ***uint32_t AlarmSubSecondMask***
- ***uint32_t AlarmDateWeekDaySel***
- ***uint8_t AlarmDateWeekDay***
- ***uint32_t Alarm***

Field Documentation

- ***RTC_TimeTypeDef RTC_AlarmTypeDef::AlarmTime***
Specifies the RTC Alarm Time members.
- ***uint32_t RTC_AlarmTypeDef::AlarmMask***
Specifies the RTC Alarm Masks. This parameter can be a value of [RTC_AlarmMask_Definitions](#).

- ***uint32_t RTC_AlarmTypeDef::AlarmSubSecondMask***
Specifies the RTC Alarm SubSeconds Masks. This parameter can be a value of [RTC_Alarm_Sub_Seconds_Masks_Definitions](#)
- ***uint32_t RTC_AlarmTypeDef::AlarmDateWeekDaySel***
Specifies the RTC Alarm is on Date or WeekDay. This parameter can be a value of [RTC_AlarmDateWeekDay_Definitions](#)
- ***uint8_t RTC_AlarmTypeDef::AlarmDateWeekDay***
Specifies the RTC Alarm Date/WeekDay. If the Alarm Date is selected, this parameter must be set to a value in the 1-31 range. If the Alarm WeekDay is selected, this parameter can be a value of [RTC_WeekDay_Definitions](#)
- ***uint32_t RTC_AlarmTypeDef::Alarm***
Specifies the alarm . This parameter can be a value of [RTC_Alarms_Definitions](#)

36.3 RTCEX Firmware driver API description

36.3.1 How to use this driver

- Enable the RTC domain access.
- Configure the RTC Prescaler (Asynchronous and Synchronous) and RTC hour format using the HAL_RTC_Init() function.

RTC Wakeup configuration

- To configure the RTC Wakeup Clock source and Counter use the HAL_RTCEX_SetWakeUpTimer() function. You can also configure the RTC Wakeup timer with interrupt mode using the HAL_RTCEX_SetWakeUpTimer_IT() function.
- To read the RTC WakeUp Counter register, use the HAL_RTCEX_GetWakeUpTimer() function.

TimeStamp configuration

- Configure the RTC_AFX trigger and enable the RTC TimeStamp using the HAL_RTCEX_SetTimeStamp() function. You can also configure the RTC TimeStamp with interrupt mode using the HAL_RTCEX_SetTimeStamp_IT() function.
- To read the RTC TimeStamp Time and Date register, use the HAL_RTCEX_GetTimeStamp() function.
- The TIMESTAMP alternate function can be mapped to RTC_AF1 (PC13).

Tamper configuration

- Enable the RTC Tamper and configure the Tamper filter count, trigger Edge or Level according to the Tamper filter (if equal to 0 Edge else Level) value, sampling frequency, precharge or discharge and Pull-UP using the HAL_RTCEX_SetTamper() function. You can configure RTC Tamper with interrupt mode using HAL_RTCEX_SetTamper_IT() function.
- The TAMPER1 alternate function can be mapped to RTC_AF1 (PC13).

Backup Data Registers configuration

- To write to the RTC Backup Data registers, use the HAL_RTCEX_BKUPWrite() function.
- To read the RTC Backup Data registers, use the HAL_RTCEX_BKUPRead() function.

36.3.2 RTC TimeStamp and Tamper functions

This section provides functions allowing to configure TimeStamp feature

This section contains the following APIs:

- [*HAL_RTCEX_SetTimeStamp\(\)*](#)
- [*HAL_RTCEX_SetTimeStamp_IT\(\)*](#)
- [*HAL_RTCEX_DeactivateTimeStamp\(\)*](#)
- [*HAL_RTCEX_GetTimeStamp\(\)*](#)
- [*HAL_RTCEX_SetTamper\(\)*](#)
- [*HAL_RTCEX_SetTamper_IT\(\)*](#)
- [*HAL_RTCEX_DeactivateTamper\(\)*](#)
- [*HAL_RTCEX_TamperTimeStampIRQHandler\(\)*](#)
- [*HAL_RTCEX_TimeStampEventCallback\(\)*](#)
- [*HAL_RTCEX_Tamper1EventCallback\(\)*](#)
- [*HAL_RTCEX_Tamper2EventCallback\(\)*](#)
- [*HAL_RTCEX_Tamper3EventCallback\(\)*](#)
- [*HAL_RTCEX_PollForTimeStampEvent\(\)*](#)
- [*HAL_RTCEX_PollForTamper1Event\(\)*](#)
- [*HAL_RTCEX_PollForTamper2Event\(\)*](#)
- [*HAL_RTCEX_PollForTamper3Event\(\)*](#)

36.3.3 RTC Wake-up functions

This section provides functions allowing to configure Wake-up feature

This section contains the following APIs:

- [*HAL_RTCEX_SetWakeUpTimer\(\)*](#)
- [*HAL_RTCEX_SetWakeUpTimer_IT\(\)*](#)
- [*HAL_RTCEX_DeactivateWakeUpTimer\(\)*](#)
- [*HAL_RTCEX_GetWakeUpTimer\(\)*](#)
- [*HAL_RTCEX_WakeUpTimerIRQHandler\(\)*](#)
- [*HAL_RTCEX_WakeUpTimerEventCallback\(\)*](#)
- [*HAL_RTCEX_PollForWakeUpTimerEvent\(\)*](#)

36.3.4 Extension Peripheral Control functions

This subsection provides functions allowing to

- Writes a data in a specified RTC Backup data register
- Read a data in a specified RTC Backup data register
- Sets the Coarse calibration parameters.
- Deactivates the Coarse calibration parameters
- Sets the Smooth calibration parameters.
- Configures the Synchronization Shift Control Settings.
- Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
- Enables the RTC reference clock detection.

- Disable the RTC reference clock detection.
- Enables the Bypass Shadow feature.
- Disables the Bypass Shadow feature.

This section contains the following APIs:

- [*HAL_RTCEX_BKUPWrite\(\)*](#)
- [*HAL_RTCEX_BKUPRead\(\)*](#)
- [*HAL_RTCEX_SetCoarseCalib\(\)*](#)
- [*HAL_RTCEX_DeactivateCoarseCalib\(\)*](#)
- [*HAL_RTCEX_SetSmoothCalib\(\)*](#)
- [*HAL_RTCEX_SetSynchroShift\(\)*](#)
- [*HAL_RTCEX_SetCalibrationOutPut\(\)*](#)
- [*HAL_RTCEX_DeactivateCalibrationOutPut\(\)*](#)
- [*HAL_RTCEX_SetRefClock\(\)*](#)
- [*HAL_RTCEX_DeactivateRefClock\(\)*](#)
- [*HAL_RTCEX_EnableBypassShadow\(\)*](#)
- [*HAL_RTCEX_DisableBypassShadow\(\)*](#)

36.3.5 Extended features functions

This section provides functions allowing to:

- RTC Alarm B callback
- RTC Poll for Alarm B request

This section contains the following APIs:

- [*HAL_RTCEX_AlarmBEventCallback\(\)*](#)
- [*HAL_RTCEX_PollForAlarmBEvent\(\)*](#)

36.3.6 HAL_RTCEX_SetTimeStamp

Function Name	HAL_StatusTypeDef HAL_RTCEX_SetTimeStamp (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge)
Function Description	Sets TimeStamp.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • TimeStampEdge: Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin. RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • This API must be called before enabling the TimeStamp feature.

36.3.7 HAL_RTCEX_SetTimeStamp_IT

Function Name	HAL_StatusTypeDef HAL_RTCEX_SetTimeStamp_IT (RTC_HandleTypeDef * hrtc, uint32_t TimeStampEdge)
Function Description	Sets TimeStamp with Interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that

contains the configuration information for RTC.

- **TimeStampEdge:** Specifies the pin edge on which the TimeStamp is activated. This parameter can be one of the following values: RTC_TIMESTAMPEDGE_RISING: the Time stamp event occurs on the rising edge of the related pin. RTC_TIMESTAMPEDGE_FALLING: the Time stamp event occurs on the falling edge of the related pin.

Return values

- HAL status

Notes

- This API must be called before enabling the TimeStamp feature.

36.3.8 HAL_RTCEX_DeactivateTimeStamp

Function Name HAL_StatusTypeDef HAL_RTCEX_DeactivateTimeStamp (RTC_HandleTypeDef * hrtc)

Function Description Deactivates TimeStamp.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- HAL status

36.3.9 HAL_RTCEX_GetTimeStamp

Function Name HAL_StatusTypeDef HAL_RTCEX_GetTimeStamp (RTC_HandleTypeDef * hrtc, RTC_TimeTypeDef * sTimeStamp, RTC_DateTypeDef * sTimeStampDate, uint32_t Format)

Function Description Gets the RTC TimeStamp value.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **sTimeStamp:** Pointer to Time structure
- **sTimeStampDate:** Pointer to Date structure
- **Format:** specifies the format of the entered parameters. This parameter can be one of the following values:
RTC_FORMAT_BIN: Binary data format
RTC_FORMAT_BCD: BCD data format

Return values

- HAL status

36.3.10 HAL_RTCEX_SetTamper

Function Name HAL_StatusTypeDef HAL_RTCEX_SetTamper (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)

Function Description Sets Tamper.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **sTamper:** Pointer to Tamper Structure.

Return values

- HAL status

Notes

- By calling this API we disable the tamper interrupt for all tampers.

36.3.11 HAL_RTCEx_SetTamper_IT

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetTamper_IT (RTC_HandleTypeDef * hrtc, RTC_TamperTypeDef * sTamper)
Function Description	Sets Tamper with interrupt.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. sTamper: Pointer to RTC Tamper.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> By calling this API we force the tamper interrupt for all tampers.

36.3.12 HAL_RTCEx_DeactivateTamper

Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateTamper (RTC_HandleTypeDef * hrtc, uint32_t Tamper)
Function Description	Deactivates Tamper.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. Tamper: Selected tamper pin. This parameter can be a value of Tamper Pins Definitions
Return values	<ul style="list-style-type: none"> HAL status

36.3.13 HAL_RTCEx_TamperTimeStampIRQHandler

Function Name	void HAL_RTCEx_TamperTimeStampIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	This function handles TimeStamp interrupt request.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> None

36.3.14 HAL_RTCEx_TimeStampEventCallback

Function Name	void HAL_RTCEx_TimeStampEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	TimeStamp callback.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> None

36.3.15 HAL_RTCEx_Tamper1EventCallback

Function Name	void HAL_RTCEx_Tamper1EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 1 callback.

Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

36.3.16 HAL_RTCEX_Tamper2EventCallback

Function Name	void HAL_RTCEX_Tamper2EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 2 callback.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • None

36.3.17 HAL_RTCEX_Tamper3EventCallback

Function Name	void HAL_RTCEX_Tamper3EventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Tamper 3 callback.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle
Return values	<ul style="list-style-type: none"> • None

36.3.18 HAL_RTCEX_PollForTimeStampEvent

Function Name	HAL_StatusTypeDef HAL_RTCEX_PollForTimeStampEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles TimeStamp polling request.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

36.3.19 HAL_RTCEX_PollForTamper1Event

Function Name	HAL_StatusTypeDef HAL_RTCEX_PollForTamper1Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper1 Polling.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

36.3.20 HAL_RTCEX_PollForTamper2Event

Function Name	HAL_StatusTypeDef HAL_RTCEX_PollForTamper2Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper2 Polling.

Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

36.3.21 HAL_RTCEx_PollForTamper3Event

Function Name	HAL_StatusTypeDef HAL_RTCEx_PollForTamper3Event (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles Tamper3 Polling.
Parameters	<ul style="list-style-type: none"> • hrtc: RTC handle • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

36.3.22 HAL_RTCEx_SetWakeUpTimer

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)
Function Description	Sets wake up timer.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • WakeUpCounter: Wake up counter • WakeUpClock: Wake up clock
Return values	<ul style="list-style-type: none"> • HAL status

36.3.23 HAL_RTCEx_SetWakeUpTimer_IT

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetWakeUpTimer_IT (RTC_HandleTypeDef * hrtc, uint32_t WakeUpCounter, uint32_t WakeUpClock)
Function Description	Sets wake up timer with interrupt.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • WakeUpCounter: Wake up counter • WakeUpClock: Wake up clock
Return values	<ul style="list-style-type: none"> • HAL status

36.3.24 HAL_RTCEx_DeactivateWakeUpTimer

Function Name	uint32_t HAL_RTCEx_DeactivateWakeUpTimer (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates wake up timer counter.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status

36.3.25 HAL_RTCEX_GetWakeUpTimer

Function Name	uint32_t HAL_RTCEX_GetWakeUpTimer (RTC_HandleTypeDef * hrtc)
Function Description	Gets wake up timer counter.
Parameters	<ul style="list-style-type: none">• hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none">• Counter value

36.3.26 HAL_RTCEX_WakeUpTimerIRQHandler

Function Name	void HAL_RTCEX_WakeUpTimerIRQHandler (RTC_HandleTypeDef * hrtc)
Function Description	This function handles Wake Up Timer interrupt request.
Parameters	<ul style="list-style-type: none">• hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none">• None

36.3.27 HAL_RTCEX_WakeUpTimerEventCallback

Function Name	void HAL_RTCEX_WakeUpTimerEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Wake Up Timer callback.
Parameters	<ul style="list-style-type: none">• hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none">• None

36.3.28 HAL_RTCEX_PollForWakeUpTimerEvent

Function Name	HAL_StatusTypeDef HAL_RTCEX_PollForWakeUpTimerEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles Wake Up Timer Polling.
Parameters	<ul style="list-style-type: none">• hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.• Timeout: Timeout duration
Return values	<ul style="list-style-type: none">• HAL status

36.3.29 HAL_RTCEX_BKUPWrite

Function Name	void HAL_RTCEX_BKUPWrite (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister, uint32_t Data)
Function Description	Writes a data in a specified RTC Backup data register.
Parameters	<ul style="list-style-type: none">• hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.• BackupRegister: RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.

- **Data:** Data to be written in the specified RTC Backup data register.

Return values

- None

36.3.30 HAL_RTCEX_BKUPRead

Function Name **uint32_t HAL_RTCEX_BKUPRead (RTC_HandleTypeDef * hrtc, uint32_t BackupRegister)**

Function Description Reads data from the specified RTC Backup data Register.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **BackupRegister:** RTC Backup data Register number. This parameter can be: RTC_BKP_DRx where x can be from 0 to 19 to specify the register.

Return values

- Read value

36.3.31 HAL_RTCEX_SetCoarseCalib

Function Name **HAL_StatusTypeDef HAL_RTCEX_SetCoarseCalib (RTC_HandleTypeDef * hrtc, uint32_t CalibSign, uint32_t Value)**

Function Description Sets the Coarse calibration parameters.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
- **CalibSign:** Specifies the sign of the coarse calibration value. This parameter can be one of the following values :
RTC_CALIBSIGN_POSITIVE: The value sign is positive
RTC_CALIBSIGN_NEGATIVE: The value sign is negative
- **Value:** value of coarse calibration expressed in ppm (coded on 5 bits).

Return values

- HAL status

Notes

- This Calibration value should be between 0 and 63 when using negative sign with a 2-ppm step.
- This Calibration value should be between 0 and 126 when using positive sign with a 4-ppm step.

36.3.32 HAL_RTCEX_DeactivateCoarseCalib

Function Name **HAL_StatusTypeDef HAL_RTCEX_DeactivateCoarseCalib (RTC_HandleTypeDef * hrtc)**

Function Description Deactivates the Coarse calibration parameters.

Parameters

- **hrtc:** pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.

Return values

- HAL status

36.3.33 HAL_RTCEX_SetSmoothCalib

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetSmoothCalib (RTC_HandleTypeDef * hrtc, uint32_t SmoothCalibPeriod, uint32_t SmoothCalibPlusPulses, uint32_t SmoothCalibMinusPulsesValue)
Function Description	Sets the Smooth calibration parameters.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • SmoothCalibPeriod: Select the Smooth Calibration Period. This parameter can be one of the following values : RTC_SMOOTHCALIB_PERIOD_32SEC: The smooth calibration period is 32s. RTC_SMOOTHCALIB_PERIOD_16SEC: The smooth calibration period is 16s. RTC_SMOOTHCALIB_PERIOD_8SEC: The smooth calibration period is 8s. • SmoothCalibPlusPulses: Select to Set or reset the CALP bit. This parameter can be one of the following values: RTC_SMOOTHCALIB_PLUSPULSES_SET: Add one RTCCLK puls every 2¹¹ pulses. RTC_SMOOTHCALIB_PLUSPULSES_RESET: No RTCCLK pulses are added. • SmoothCalibMinusPulsesValue: Select the value of CALM[8:0] bits. This parameter can be any value from 0 to 0x000001FF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • To deactivate the smooth calibration, the field SmoothCalibPlusPulses must be equal to SMOOTHCALIB_PLUSPULSES_RESET and the field SmoothCalibMinusPulsesValue must be equal to 0.

36.3.34 HAL_RTCEx_SetSynchroShift

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetSynchroShift (RTC_HandleTypeDef * hrtc, uint32_t ShiftAdd1S, uint32_t ShiftSubFS)
Function Description	Configures the Synchronization Shift Control Settings.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • ShiftAdd1S: Select to add or not 1 second to the time calendar. This parameter can be one of the following values : RTC_SHIFTADD1S_SET: Add one second to the clock calendar. RTC_SHIFTADD1S_RESET: No effect. • ShiftSubFS: Select the number of Second Fractions to substitute. This parameter can be any value from 0 to 0x7FFF.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When REFCKON is set, firmware must not write to Shift control register.

36.3.35 HAL_RTCEx_SetCalibrationOutPut

Function Name	HAL_StatusTypeDef HAL_RTCEx_SetCalibrationOutPut (RTC_HandleTypeDef * hrtc, uint32_t CalibOutput)
Function Description	Configures the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. • CalibOutput: : Select the Calibration output Selection . This parameter can be one of the following values: RTC_CALIBOUTPUT_512HZ: A signal has a regular waveform at 512Hz.RTC_CALIBOUTPUT_1HZ: A signal has a regular waveform at 1Hz.
Return values	<ul style="list-style-type: none"> • HAL status
36.3.36 HAL_RTCEx_DeactivateCalibrationOutPut	
Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateCalibrationOutPut (RTC_HandleTypeDef * hrtc)
Function Description	Deactivates the Calibration Pinout (RTC_CALIB) Selection (1Hz or 512Hz).
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
36.3.37 HAL_RTCEx_SetRefClock	
Function Name	HAL_StatusTypeDef HAL_RTCEx_SetRefClock (RTC_HandleTypeDef * hrtc)
Function Description	Enables the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
36.3.38 HAL_RTCEx_DeactivateRefClock	
Function Name	HAL_StatusTypeDef HAL_RTCEx_DeactivateRefClock (RTC_HandleTypeDef * hrtc)
Function Description	Disable the RTC reference clock detection.
Parameters	<ul style="list-style-type: none"> • hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> • HAL status
36.3.39 HAL_RTCEx_EnableBypassShadow	
Function Name	HAL_StatusTypeDef HAL_RTCEx_EnableBypassShadow (RTC_HandleTypeDef * hrtc)
Function Description	Enables the Bypass Shadow feature.

Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

36.3.40 HAL_RTCEX_DisableBypassShadow

Function Name	HAL_StatusTypeDef HAL_RTCEX_DisableBypassShadow (RTC_HandleTypeDef * hrtc)
Function Description	Disables the Bypass Shadow feature.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> HAL status
Notes	<ul style="list-style-type: none"> When the Bypass Shadow is enabled the calendar value are taken directly from the Calendar counter.

36.3.41 HAL_RTCEX_AlarmBEventCallback

Function Name	void HAL_RTCEX_AlarmBEventCallback (RTC_HandleTypeDef * hrtc)
Function Description	Alarm B callback.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC.
Return values	<ul style="list-style-type: none"> None

36.3.42 HAL_RTCEX_PollForAlarmBEvent

Function Name	HAL_StatusTypeDef HAL_RTCEX_PollForAlarmBEvent (RTC_HandleTypeDef * hrtc, uint32_t Timeout)
Function Description	This function handles AlarmB Polling request.
Parameters	<ul style="list-style-type: none"> hrtc: pointer to a RTC_HandleTypeDef structure that contains the configuration information for RTC. Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> HAL status

36.4 RTCEX Firmware driver defines

36.4.1 RTCEX

Add 1 Second Parameter Definitions

RTC_SHIFTADD1S_RESET

RTC_SHIFTADD1S_SET

IS_RTC_SHIFT_ADD1S

Backup Registers Definitions

RTC_BKP_DR0
RTC_BKP_DR1
RTC_BKP_DR2
RTC_BKP_DR3
RTC_BKP_DR4
RTC_BKP_DR5
RTC_BKP_DR6
RTC_BKP_DR7
RTC_BKP_DR8
RTC_BKP_DR9
RTC_BKP_DR10
RTC_BKP_DR11
RTC_BKP_DR12
RTC_BKP_DR13
RTC_BKP_DR14
RTC_BKP_DR15
RTC_BKP_DR16
RTC_BKP_DR17
RTC_BKP_DR18
RTC_BKP_DR19
RTC_BKP_DR20
RTC_BKP_DR21
RTC_BKP_DR22
RTC_BKP_DR23
RTC_BKP_DR24
RTC_BKP_DR25
RTC_BKP_DR26
RTC_BKP_DR27
RTC_BKP_DR28
RTC_BKP_DR29
RTC_BKP_DR30
RTC_BKP_DR31
IS_RTC_BKP

Calib Output Selection Definitions

RTC_CALIBOUTPUT_512HZ
RTC_CALIBOUTPUT_1HZ

IS_RTC_CALIB_OUTPUT

Digital Calibration Definitions

RTC_CALIBSIGN_POSITIVE

RTC_CALIBSIGN_NEGATIVE

IS_RTC_CALIB_SIGN

IS_RTC_CALIB_VALUE

RTCEX Exported Macros

__HAL_RTC_WAKEUPTIMER_ENABLE

Description:

- Enable the RTC WakeUp Timer peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_ENABLE

Description:

- Enable the RTC TimeStamp peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

__HAL_RTC_WAKEUPTIMER_DISABLE

Description:

- Disable the RTC WakeUp Timer peripheral.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

__HAL_RTC_TIMESTAMP_DISABLE

Description:

- Disable the RTC TimeStamp peripheral.

__HAL_RTC_COARSE_CALIB_ENABLE

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

Description:

- Enable the Coarse calibration process.

__HAL_RTC_COARSE_CALIB_DISABLE

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

Description:

- Disable the Coarse calibration process.

__HAL_RTC_CALIBRATION_OUTPUT_ENABLE

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

Description:

- Enable the RTC calibration output.

__HAL_RTC_CALIBRATION_OUTPUT_DISABLE

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

- None

Description:

- Disable the calibration output.

Parameters:

- __HANDLE__: specifies the RTC handle.

Return value:

`__HAL_RTC_CLOCKREF_DETECTION_ENABLE`

- None

Description:

- Enable the clock reference detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_CLOCKREF_DETECTION_DISABLE`

Description:

- Disable the clock reference detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

`__HAL_RTC_TIMESTAMP_ENABLE_IT`

Description:

- Enable the RTC TimeStamp interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
 - `RTC_IT_TS`: TimeStamp interrupt

Return value:

- None

`__HAL_RTC_WAKEUPTIMER_ENABLE_IT`

Description:

- Enable the RTC WakeUpTimer interrupt.

Parameters:

- `__HANDLE__`:

__HAL_RTC_TIMESTAMP_DISABLE_IT

specifies the RTC handle.

- **__INTERRUPT__**: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
 - **RTC_IT_WUT**: WakeUpTimer A interrupt

Return value:

- None

Description:

- Disable the RTC TimeStamp interrupt.

Parameters:

- **__HANDLE__**: specifies the RTC handle.
- **__INTERRUPT__**: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
 - **RTC_IT_TS**: TimeStamp interrupt

Return value:

- None

Description:

- Disable the RTC WakeUpTimer interrupt.

Parameters:

- **__HANDLE__**: specifies the RTC handle.
- **__INTERRUPT__**: specifies the RTC WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:
 - **RTC_IT_WUT**: WakeUpTimer A

interrupt

`__HAL_RTC_TAMPER1_ENABLE`**Return value:**

- None

Description:

- Enable the RTC Tamper1 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Disable the RTC Tamper1 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Enable the RTC Tamper2 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Disable the RTC Tamper2 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

`__HAL_RTC_TAMPER1_DISABLE``__HAL_RTC_TAMPER2_ENABLE``__HAL_RTC_TAMPER2_DISABLE`

`__HAL_RTC_TAMPER3_ENABLE`

Return value:

- None

Description:

- Enable the RTC Tamper3 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Disable the RTC Tamper3 input detection.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

Return value:

- None

Description:

- Check whether the specified RTC Tamper interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Tamper interrupt sources to be enabled or disabled. This parameter can be:
 - `RTC_IT_TAMP1`
 - `RTC_IT_TAMP2`
 - `RTC_IT_TAMP3`

Return value:

- None

Description:

- Enable the RTC

`__HAL_RTC_TAMPER3_DISABLE`

`__HAL_RTC_TAMPER_GET_IT`

`__HAL_RTC_TAMPER_ENABLE_IT`

Tamper interrupt.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Tamper interrupt sources to be enabled. This parameter can be any combination of the following values:
 - `RTC_IT_TAMP1`: Tamper1 interrupt
 - `RTC_IT_TAMP2`: Tamper2 interrupt (*)
 - `RTC_IT_TAMP3`: Tamper3 interrupt (*)

Return value:

- None

Notes:

- (*) Available only on devices
STM32L100xBA,
STM32L151xBA,
STM32L152xBA,
STM32L100xC,
STM32L151xC,
STM32L152xC,
STM32L162xC,
STM32L151xCA,
STM32L151xD,
STM32L152xCA,
STM32L152xD,
STM32L162xCA,
STM32L162xD,
STM32L151xE,
STM32L152xE,
STM32L162xE,
STM32L151DX,
STM32L152DX,
STM32L162DX

Description:

- Disable the RTC Tamper interrupt.

Parameters:

- `__HANDLE__`:

`__HAL_RTC_TAMPER_DISABLE_IT`

specifies the RTC handle.

- **__INTERRUPT__**: specifies the RTC Tamper interrupt sources to be disabled. This parameter can be any combination of the following values:
 - **RTC_IT_TAMP1**: Tamper1 interrupt
 - **RTC_IT_TAMP2**: Tamper2 interrupt (*)
 - **RTC_IT_TAMP3**: Tamper3 interrupt (*)

Return value:

- None

Notes:

- (*) Available only on devices
STM32L100xBA,
STM32L151xBA,
STM32L152xBA,
STM32L100xC,
STM32L151xC,
STM32L152xC,
STM32L162xC,
STM32L151xCA,
STM32L151xD,
STM32L152xCA,
STM32L152xD,
STM32L162xCA,
STM32L162xD,
STM32L151xE,
STM32L152xE,
STM32L162xE,
STM32L151DX,
STM32L152DX,
STM32L162DX

__HAL_RTC_TAMPER_GET_IT_SOURCE

Description:

- Check whether the specified RTC Tamper interrupt has been enabled or not.

Parameters:

- **__HANDLE__**: specifies the RTC

handle.

- **__INTERRUPT__**: specifies the RTC Tamper interrupt source to check. This parameter can be:
 - **RTC_IT_TAMP1**: Tamper1 interrupt
 - **RTC_IT_TAMP2**: Tamper2 interrupt (*)
 - **RTC_IT_TAMP3**: Tamper3 interrupt (*)

Return value:

- None

Notes:

- (*) Available only on devices
STM32L100xBA,
STM32L151xBA,
STM32L152xBA,
STM32L100xC,
STM32L151xC,
STM32L152xC,
STM32L162xC,
STM32L151xCA,
STM32L151xD,
STM32L152xCA,
STM32L152xD,
STM32L162xCA,
STM32L162xD,
STM32L151xE,
STM32L152xE,
STM32L162xE,
STM32L151DX,
STM32L152DX,
STM32L162DX

__HAL_RTC_WAKEUPTIMER_GET_IT**Description:**

- Check whether the specified RTC WakeUpTimer interrupt has occurred or not.

Parameters:

- **__HANDLE__**: specifies the RTC handle.
- **__FLAG__**: specifies the RTC

__HAL_RTC_WAKEUPTIMER_GET_IT_SOURCE

WakeUpTimer interrupt sources to be enabled or disabled. This parameter can be:

- RTC_IT_WUT: WakeUpTimer A interrupt

Return value:

- None

Description:

- Check whether the specified RTC Wake Up timer interrupt has been enabled or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC Wake Up timer interrupt sources to check. This parameter can be:
 - RTC_IT_WUT: WakeUpTimer interrupt

Return value:

- None

__HAL_RTC_TIMESTAMP_GET_IT

Description:

- Check whether the specified RTC TimeStamp interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the RTC handle.
- __INTERRUPT__: specifies the RTC TimeStamp interrupt sources to be enabled or disabled. This parameter can be:
 - RTC_IT_TS: TimeStamp interrupt

`__HAL_RTC_TIMESTAMP_GET_IT_SOURCE`**Return value:**

- None

Description:

- Check whether the specified RTC Time Stamp interrupt has been enabled or not.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__INTERRUPT__`: specifies the RTC Time Stamp interrupt source to check. This parameter can be:
 - `RTC_IT_TS`: TimeStamp interrupt

Return value:

- None

Description:

- Get the selected RTC TimeStamp's flag status.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC TimeStamp Flag sources to be enabled or disabled. This parameter can be:
 - `RTC_FLAG_TSF`
 - `RTC_FLAG_TS_OVF`

Return value:

- None

Description:

- Get the selected RTC WakeUpTimer's flag status.

Parameters:

- `__HANDLE__`:

`__HAL_RTC_TIMESTAMP_GET_FLAG``__HAL_RTC_WAKEUPTIMER_GET_FLAG`

`__HAL_RTC_TAMPER_GET_FLAG`

specifies the RTC handle.

- `__FLAG__`: specifies the RTC WakeUpTimer Flag sources to be enabled or disabled. This parameter can be:
 - `RTC_FLAG_WUTF`
 - `RTC_FLAG_WUTWF`

Return value:

- None

Description:

- Get the selected RTC Tamper's flag status.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
 - `RTC_FLAG_TAMP1F`
 - `RTC_FLAG_TAMP2F (*)`
 - `RTC_FLAG_TAMP3F (*)`

Return value:

- None

Notes:

- (*) Available only on devices
 STM32L100xBA,
 STM32L151xBA,
 STM32L152xBA,
 STM32L100xC,
 STM32L151xC,
 STM32L152xC,
 STM32L162xC,
 STM32L151xCA,
 STM32L151xD,
 STM32L152xCA,
 STM32L152xD,
 STM32L162xCA,
 STM32L162xD,

STM32L151xE,
STM32L152xE,
STM32L162xE,
STM32L151xDX,
STM32L152xDX,
STM32L162xDX

__HAL_RTC_SHIFT_GET_FLAG

Description:

- Get the selected RTC shift operation's flag status.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC shift operation Flag is pending or not. This parameter can be:
 - `RTC_FLAG_SHPF`

Return value:

- None

__HAL_RTC_TIMESTAMP_CLEAR_FLAG

Description:

- Clear the RTC Time Stamp's pending flags.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Alarm Flag sources to be enabled or disabled. This parameter can be:
 - `RTC_FLAG_TSF`

Return value:

- None

__HAL_RTC_TAMPER_CLEAR_FLAG

Description:

- Clear the RTC Tamper's pending flags.

Parameters:

- `__HANDLE__`: specifies the RTC handle.

- `__FLAG__`: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
 - `RTC_FLAG_TAMP1F`
 - `RTC_FLAG_TAMP2F (*)`
 - `RTC_FLAG_TAMP3F (*)`

Return value:

- None

Notes:

- (*) Available only on devices
STM32L100xBA,
STM32L151xBA,
STM32L152xBA,
STM32L100xC,
STM32L151xC,
STM32L152xC,
STM32L162xC,
STM32L151xCA,
STM32L151xD,
STM32L152xCA,
STM32L152xD,
STM32L162xCA,
STM32L162xD,
STM32L151xE,
STM32L152xE,
STM32L162xE,
STM32L151DX,
STM32L152DX,
STM32L162DX

`__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG`

Description:

- Clear the RTC Wake Up timer's pending flags.

Parameters:

- `__HANDLE__`: specifies the RTC handle.
- `__FLAG__`: specifies the RTC Tamper Flag sources to be enabled or disabled. This parameter can be:
 - `RTC_FLAG_WUTF`

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_IT

Return value:

- None

Description:

- Enable interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None

Description:

- Disable interrupt on the RTC WakeUp Timer associated Exti line.

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_IT

Return value:

- None

Description:

- Enable event on the RTC WakeUp Timer associated Exti line.

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_EVENT

Return value:

- None.

Description:

- Disable event on the RTC WakeUp Timer associated Exti line.

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_EVENT

Return value:

- None.

Description:

- Enable falling edge trigger on the RTC WakeUp Timer associated Exti line.

__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_FALLING_EDGE

Return value:

- None.

Description:

- Disable falling edge trigger on the RTC WakeUp Timer associated Exti line.

__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_FALLING_EDGE

Return value:

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_EDGE`

- None.

Description:

- Enable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_EDGE`

Description:

- Disable rising edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_ENABLE_RISING_FALLING_EDGE`

Description:

- Enable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_DISABLE_RISING_FALLING_EDGE`

Description:

- Disable rising & falling edge trigger on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

`__HAL_RTC_WAKEUPTIMER_EXTI_GET_FLAG`

Description:

- Check whether the RTC WakeUp Timer associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

`__HAL_RTC_WAKEUPTIMER_EXTI_CLEAR_FLAG`

Description:

- Clear the RTC WakeUp Timer associated Exti line flag.

__HAL_RTC_WAKEUPTIMER_EXTI_GENERATE_SWIT

Return value:

- None.

Description:

- Generate a Software interrupt on the RTC WakeUp Timer associated Exti line.

Return value:

- None.

Description:

- Enable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

Description:

- Disable interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None

Description:

- Enable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

Description:

- Disable event on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

Description:

- Enable falling edge trigger on the RTC Tamper and Timestamp

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_IT

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_IT

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_EVENT

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_EVENT

__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_FALLING_EDGE

associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_FALLING_EDGE`

Description:

- Disable falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_EDGE`

Description:

- Enable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_EDGE`

Description:

- Disable rising edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_ENABLE_RISING_FALLING_EDGE`

Description:

- Enable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_DISABLE_RISING_FALLING_EDGE`

Description:

- Disable rising & falling edge trigger on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GET_FLAG`**Description:**

- Check whether the RTC Tamper and Timestamp associated Exti line interrupt flag is set or not.

Return value:

- Line: Status.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_CLEAR_FLAG`**Description:**

- Clear the RTC Tamper and Timestamp associated Exti line flag.

Return value:

- None.

`__HAL_RTC_TAMPER_TIMESTAMP_EXTI_GENERATE_SWIT`**Description:**

- Generate a Software interrupt on the RTC Tamper and Timestamp associated Exti line.

Return value:

- None.

Output selection Definitions`RTC_OUTPUT_DISABLE``RTC_OUTPUT_ALARMA``RTC_OUTPUT_ALARMB``RTC_OUTPUT_WAKEUP``IS_RTC_OUTPUT`**RTCEX Private Constants**

`RTC_EXTI_LINE_TAMPER_TIMESTAMP_EVENT` External interrupt line 19 Connected to the RTC Tamper and Time Stamp events

`RTC_EXTI_LINE_WAKEUPTIMER_EVENT` External interrupt line 20 Connected to the RTC Wakeup event

Smooth Calib Minus Pulses Definitions`IS_RTC_SMOOTH_CALIB_MINUS`**Smooth Calib Period Definitions**

`RTC_SMOOTHCALIB_PERIOD_32SEC` If RTCCLK = 32768 Hz, Smooth calibration period is 32s, else 2exp20 RTCCLK seconds

RTC_SMOOTHCALIB_PERIOD_16SEC If RTCCLK = 32768 Hz, Smooth calibration period is 16s, else 2exp19 RTCCLK seconds

RTC_SMOOTHCALIB_PERIOD_8SEC If RTCCLK = 32768 Hz, Smooth calibration period is 8s, else 2exp18 RTCCLK seconds

IS_RTC_SMOOTH_CALIB_PERIOD

Smooth Calib Plus Pulses Definitions

RTC_SMOOTHCALIB_PLUSPULSES_SET The number of RTCCLK pulses added during a X -second window = Y - CALM[8:0] with Y = 512, 256, 128 when X = 32, 16, 8

RTC_SMOOTHCALIB_PLUSPULSES_RESET The number of RTCCLK pulses subttited during a 32-second window = CALM[8:0]

IS_RTC_SMOOTH_CALIB_PLUS

Substract Fraction Of Second Value

IS_RTC_SHIFT_SUBFS

Tamper Filter Definitions

RTC_TAMPERFILTER_DISABLE Tamper filter is disabled

RTC_TAMPERFILTER_2SAMPLE Tamper is activated after 2 consecutive samples at the active level

RTC_TAMPERFILTER_4SAMPLE Tamper is activated after 4 consecutive samples at the active level

RTC_TAMPERFILTER_8SAMPLE Tamper is activated after 8 consecutive samples at the active level.

IS_RTC_TAMPER_FILTER

Tamper Pins Definitions

RTC_TAMPER_1

RTC_TAMPER_2

RTC_TAMPER_3

IS_RTC_TAMPER

Tamper Pin Precharge Duration

RTC_TAMPERPRECHARGEDURATION_1RTCCLK Tamper pins are pre-charged before sampling during 1 RTCCLK cycle

RTC_TAMPERPRECHARGEDURATION_2RTCCLK Tamper pins are pre-charged before sampling during 2 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_4RTCCLK Tamper pins are pre-charged before sampling during 4 RTCCLK cycles

RTC_TAMPERPRECHARGEDURATION_8RTCCLK Tamper pins are pre-charged before sampling during 8 RTCCLK cycles

IS_RTC_TAMPER_PRECHARGE_DURATION
Tamper Pull-Up Definitions

RTC_TAMPER_PULLUP_ENABLE TimeStamp on Tamper Detection event saved

RTC_TAMPER_PULLUP_DISABLE TimeStamp on Tamper Detection event is not saved

IS_RTC_TAMPER_PULLUP_STATE

Tamper Sampling FrequenciesRTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV32768 Each of the tamper inputs are sampled with a frequency = $RTCCLK / 32768$ RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV16384 Each of the tamper inputs are sampled with a frequency = $RTCCLK / 16384$ RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV8192 Each of the tamper inputs are sampled with a frequency = $RTCCLK / 8192$ RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV4096 Each of the tamper inputs are sampled with a frequency = $RTCCLK / 4096$ RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV2048 Each of the tamper inputs are sampled with a frequency = $RTCCLK / 2048$ RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV1024 Each of the tamper inputs are sampled with a frequency = $RTCCLK / 1024$ RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV512 Each of the tamper inputs are sampled with a frequency = $RTCCLK / 512$ RTC_TAMPERSAMPLINGFREQ_RTCCLK_DIV256 Each of the tamper inputs are sampled with a frequency = $RTCCLK / 256$

IS_RTC_TAMPER_SAMPLING_FREQ

TimeStampOnTamperDetection Definitions

RTC_TIMESTAMPONTAMPERDETECTION_ENABLE TimeStamp on Tamper Detection event saved

RTC_TIMESTAMPONTAMPERDETECTION_DISABLE TimeStamp on Tamper Detection event is not saved

IS_RTC_TAMPER_TIMESTAMPONTAMPER_DETECTION

Tamper Trigger Definitions

RTC_TAMPERTRIGGER_RISINGEDGE

RTC_TAMPERTRIGGER_FALLINGEDGE

RTC_TAMPERTRIGGER_LOWLEVEL

RTC_TAMPERTRIGGER_HIGHLEVEL

IS_RTC_TAMPER_TRIGGER

Time Stamp Edges Definitions

RTC_TIMESTAMPEDGE_RISING

RTC_TIMESTAMPEDGE_FALLING

IS_TIMESTAMP_EDGE

Wakeup Timer Definitions

RTC_WAKEUPCLOCK_RTCCLK_DIV16

RTC_WAKEUPCLOCK_RTCCLK_DIV8

RTC_WAKEUPCLOCK_RTCCLK_DIV4

RTC_WAKEUPCLOCK_RTCCLK_DIV2

RTC_WAKEUPCLOCK_CK_SPRE_16BITS

RTC_WAKEUPCLOCK_CK_SPRE_17BITS

IS_RTC_WAKEUP_CLOCK

IS_RTC_WAKEUP_COUNTER

37 HAL SD Generic Driver

37.1 HAL SD Generic Driver

37.2 SD Firmware driver registers structures

37.2.1 SD_HandleTypeDef

Data Fields

- ***SD_TypeDef * Instance***
- ***SD_InitTypeDef Init***
- ***HAL_LockTypeDef Lock***
- ***uint32_t CardType***
- ***uint32_t RCA***
- ***uint32_t CSD***
- ***uint32_t CID***
- ***__IO uint32_t SdTransferCplt***
- ***__IO uint32_t SdTransferErr***
- ***__IO uint32_t DmaTransferCplt***
- ***__IO uint32_t SdOperation***
- ***DMA_HandleTypeDef * hdmarx***
- ***DMA_HandleTypeDef * hdmatx***

Field Documentation

- ***SD_TypeDef* SD_HandleTypeDef::Instance***
SDIO register base address
- ***SD_InitTypeDef SD_HandleTypeDef::Init***
SD required parameters
- ***HAL_LockTypeDef SD_HandleTypeDef::Lock***
SD locking object
- ***uint32_t SD_HandleTypeDef::CardType***
SD card type
- ***uint32_t SD_HandleTypeDef::RCA***
SD relative card address
- ***uint32_t SD_HandleTypeDef::CSD[4]***
SD card specific data table
- ***uint32_t SD_HandleTypeDef::CID[4]***
SD card identification number table
- ***__IO uint32_t SD_HandleTypeDef::SdTransferCplt***
SD transfer complete flag in non blocking mode
- ***__IO uint32_t SD_HandleTypeDef::SdTransferErr***
SD transfer error flag in non blocking mode
- ***__IO uint32_t SD_HandleTypeDef::DmaTransferCplt***
SD DMA transfer complete flag
- ***__IO uint32_t SD_HandleTypeDef::SdOperation***
SD transfer operation (read/write)

- ***DMA_HandleTypeDef* SD_HandleTypeDef::hdmarx***
SD Rx DMA handle parameters
- ***DMA_HandleTypeDef* SD_HandleTypeDef::hdmatx***
SD Tx DMA handle parameters

37.2.2 HAL_SD_CSDTypedef

Data Fields

- ***__IO uint8_t CSDStruct***
- ***__IO uint8_t SysSpecVersion***
- ***__IO uint8_t Reserved1***
- ***__IO uint8_t TAAC***
- ***__IO uint8_t NSAC***
- ***__IO uint8_t MaxBusClkFrec***
- ***__IO uint16_t CardComdClasses***
- ***__IO uint8_t RdBlockLen***
- ***__IO uint8_t PartBlockRead***
- ***__IO uint8_t WrBlockMisalign***
- ***__IO uint8_t RdBlockMisalign***
- ***__IO uint8_t DSRImpl***
- ***__IO uint8_t Reserved2***
- ***__IO uint32_t DeviceSize***
- ***__IO uint8_t MaxRdCurrentVDDMin***
- ***__IO uint8_t MaxRdCurrentVDDMax***
- ***__IO uint8_t MaxWrCurrentVDDMin***
- ***__IO uint8_t MaxWrCurrentVDDMax***
- ***__IO uint8_t DeviceSizeMul***
- ***__IO uint8_t EraseGrSize***
- ***__IO uint8_t EraseGrMul***
- ***__IO uint8_t WrProtectGrSize***
- ***__IO uint8_t WrProtectGrEnable***
- ***__IO uint8_t ManDeflECC***
- ***__IO uint8_t WrSpeedFact***
- ***__IO uint8_t MaxWrBlockLen***
- ***__IO uint8_t WriteBlockPaPartial***
- ***__IO uint8_t Reserved3***
- ***__IO uint8_t ContentProtectAppli***
- ***__IO uint8_t FileFormatGrouop***
- ***__IO uint8_t CopyFlag***
- ***__IO uint8_t PermWrProtect***
- ***__IO uint8_t TempWrProtect***
- ***__IO uint8_t FileFormat***
- ***__IO uint8_t ECC***
- ***__IO uint8_t CSD_CRC***
- ***__IO uint8_t Reserved4***

Field Documentation

- **__IO uint8_t HAL_SD_CSDTypeDef::CSDStruct**
CSD structure
- **__IO uint8_t HAL_SD_CSDTypeDef::SysSpecVersion**
System specification version
- **__IO uint8_t HAL_SD_CSDTypeDef::Reserved1**
Reserved
- **__IO uint8_t HAL_SD_CSDTypeDef::TAAC**
Data read access time 1
- **__IO uint8_t HAL_SD_CSDTypeDef::NSAC**
Data read access time 2 in CLK cycles
- **__IO uint8_t HAL_SD_CSDTypeDef::MaxBusClkFrec**
Max. bus clock frequency
- **__IO uint16_t HAL_SD_CSDTypeDef::CardComdClasses**
Card command classes
- **__IO uint8_t HAL_SD_CSDTypeDef::RdBlockLen**
Max. read data block length
- **__IO uint8_t HAL_SD_CSDTypeDef::PartBlockRead**
Partial blocks for read allowed
- **__IO uint8_t HAL_SD_CSDTypeDef::WrBlockMisalign**
Write block misalignment
- **__IO uint8_t HAL_SD_CSDTypeDef::RdBlockMisalign**
Read block misalignment
- **__IO uint8_t HAL_SD_CSDTypeDef::DSRImpl**
DSR implemented
- **__IO uint8_t HAL_SD_CSDTypeDef::Reserved2**
Reserved
- **__IO uint32_t HAL_SD_CSDTypeDef::DeviceSize**
Device Size
- **__IO uint8_t HAL_SD_CSDTypeDef::MaxRdCurrentVDDMin**
Max. read current @ VDD min
- **__IO uint8_t HAL_SD_CSDTypeDef::MaxRdCurrentVDDMax**
Max. read current @ VDD max
- **__IO uint8_t HAL_SD_CSDTypeDef::MaxWrCurrentVDDMin**
Max. write current @ VDD min
- **__IO uint8_t HAL_SD_CSDTypeDef::MaxWrCurrentVDDMax**
Max. write current @ VDD max
- **__IO uint8_t HAL_SD_CSDTypeDef::DeviceSizeMul**
Device size multiplier
- **__IO uint8_t HAL_SD_CSDTypeDef::EraseGrSize**
Erase group size
- **__IO uint8_t HAL_SD_CSDTypeDef::EraseGrMul**
Erase group size multiplier
- **__IO uint8_t HAL_SD_CSDTypeDef::WrProtectGrSize**
Write protect group size
- **__IO uint8_t HAL_SD_CSDTypeDef::WrProtectGrEnable**
Write protect group enable
- **__IO uint8_t HAL_SD_CSDTypeDef::ManDeflECC**
Manufacturer default ECC
- **__IO uint8_t HAL_SD_CSDTypeDef::WrSpeedFact**
Write speed factor
- **__IO uint8_t HAL_SD_CSDTypeDef::MaxWrBlockLen**
Max. write data block length
- **__IO uint8_t HAL_SD_CSDTypeDef::WriteBlockPaPartial**
Partial blocks for write allowed

- **__IO uint8_t HAL_SD_CSDTypedef::Reserved3**
Reserved
- **__IO uint8_t HAL_SD_CSDTypedef::ContentProtectAppli**
Content protection application
- **__IO uint8_t HAL_SD_CSDTypedef::FileFormatGroup**
File format group
- **__IO uint8_t HAL_SD_CSDTypedef::CopyFlag**
Copy flag (OTP)
- **__IO uint8_t HAL_SD_CSDTypedef::PermWrProtect**
Permanent write protection
- **__IO uint8_t HAL_SD_CSDTypedef::TempWrProtect**
Temporary write protection
- **__IO uint8_t HAL_SD_CSDTypedef::FileFormat**
File format
- **__IO uint8_t HAL_SD_CSDTypedef::ECC**
ECC code
- **__IO uint8_t HAL_SD_CSDTypedef::CSD_CRC**
CSD CRC
- **__IO uint8_t HAL_SD_CSDTypedef::Reserved4**
Always 1

37.2.3 HAL_SD_CIDTypedef

Data Fields

- **__IO uint8_t ManufacturerID**
- **__IO uint16_t OEM_AppliID**
- **__IO uint32_t ProdName1**
- **__IO uint8_t ProdName2**
- **__IO uint8_t ProdRev**
- **__IO uint32_t ProdSN**
- **__IO uint8_t Reserved1**
- **__IO uint16_t ManufactDate**
- **__IO uint8_t CID_CRC**
- **__IO uint8_t Reserved2**

Field Documentation

- **__IO uint8_t HAL_SD_CIDTypedef::ManufacturerID**
Manufacturer ID
- **__IO uint16_t HAL_SD_CIDTypedef::OEM_AppliID**
OEM/Application ID
- **__IO uint32_t HAL_SD_CIDTypedef::ProdName1**
Product Name part1
- **__IO uint8_t HAL_SD_CIDTypedef::ProdName2**
Product Name part2
- **__IO uint8_t HAL_SD_CIDTypedef::ProdRev**
Product Revision
- **__IO uint32_t HAL_SD_CIDTypedef::ProdSN**
Product Serial Number

- **__IO uint8_t HAL_SD_CIDTypeDef::Reserved1**
Reserved1
- **__IO uint16_t HAL_SD_CIDTypeDef::ManufactDate**
Manufacturing Date
- **__IO uint8_t HAL_SD_CIDTypeDef::CID_CRC**
CID CRC
- **__IO uint8_t HAL_SD_CIDTypeDef::Reserved2**
Always 1

37.2.4 HAL_SD_CardStatusTypeDef

Data Fields

- **__IO uint8_t DAT_BUS_WIDTH**
- **__IO uint8_t SECURED_MODE**
- **__IO uint16_t SD_CARD_TYPE**
- **__IO uint32_t SIZE_OF_PROTECTED_AREA**
- **__IO uint8_t SPEED_CLASS**
- **__IO uint8_t PERFORMANCE_MOVE**
- **__IO uint8_t AU_SIZE**
- **__IO uint16_t ERASE_SIZE**
- **__IO uint8_t ERASE_TIMEOUT**
- **__IO uint8_t ERASE_OFFSET**

Field Documentation

- **__IO uint8_t HAL_SD_CardStatusTypeDef::DAT_BUS_WIDTH**
Shows the currently defined data bus width
- **__IO uint8_t HAL_SD_CardStatusTypeDef::SECURED_MODE**
Card is in secured mode of operation
- **__IO uint16_t HAL_SD_CardStatusTypeDef::SD_CARD_TYPE**
Carries information about card type
- **__IO uint32_t HAL_SD_CardStatusTypeDef::SIZE_OF_PROTECTED_AREA**
Carries information about the capacity of protected area
- **__IO uint8_t HAL_SD_CardStatusTypeDef::SPEED_CLASS**
Carries information about the speed class of the card
- **__IO uint8_t HAL_SD_CardStatusTypeDef::PERFORMANCE_MOVE**
Carries information about the card's performance move
- **__IO uint8_t HAL_SD_CardStatusTypeDef::AU_SIZE**
Carries information about the card's allocation unit size
- **__IO uint16_t HAL_SD_CardStatusTypeDef::ERASE_SIZE**
Determines the number of AUs to be erased in one operation
- **__IO uint8_t HAL_SD_CardStatusTypeDef::ERASE_TIMEOUT**
Determines the timeout for any number of AU erase
- **__IO uint8_t HAL_SD_CardStatusTypeDef::ERASE_OFFSET**
Carries information about the erase offset

37.2.5 HAL_SD_CardInfoTypeDef

Data Fields

- *HAL_SD_CSDTypeDef* *SD_csd*
- *HAL_SD_CIDTypeDef* *SD_cid*
- *uint64_t* *CardCapacity*
- *uint32_t* *CardBlockSize*
- *uint16_t* *RCA*
- *uint8_t* *CardType*

Field Documentation

- *HAL_SD_CSDTypeDef* *HAL_SD_CardInfoTypeDef::SD_csd*
SD card specific data register
- *HAL_SD_CIDTypeDef* *HAL_SD_CardInfoTypeDef::SD_cid*
SD card identification number register
- *uint64_t* *HAL_SD_CardInfoTypeDef::CardCapacity*
Card capacity
- *uint32_t* *HAL_SD_CardInfoTypeDef::CardBlockSize*
Card block size
- *uint16_t* *HAL_SD_CardInfoTypeDef::RCA*
SD relative card address
- *uint8_t* *HAL_SD_CardInfoTypeDef::CardType*
SD card type

37.3 SD Firmware driver API description

37.3.1 How to use this driver

This driver implements a high level communication layer for read and write from/to this memory. The needed STM32 hardware resources (SDIO and GPIO) are performed by the user in *HAL_SD_MspInit()* function (MSP layer). Basically, the MSP layer configuration should be the same as we provide in the examples. You can easily tailor this configuration according to hardware resources.

This driver is a generic layered driver for SDIO memories which uses the HAL SDIO driver functions to interface with SD and uSD cards devices. It is used as follows:

1. Initialize the SDIO low level resources by implement the *HAL_SD_MspInit()* API:
 - a. Enable the SDIO interface clock using *__HAL_RCC_SDIO_CLK_ENABLE()*;
 - b. SDIO pins configuration for SD card
 - Enable the clock for the SDIO GPIOs using the functions *__HAL_RCC_GPIOx_CLK_ENABLE()*;
 - Configure these SDIO pins as alternate function pull-up using *HAL_GPIO_Init()* and according to your pin assignment;
 - c. DMA Configuration if you need to use DMA process (*HAL_SD_ReadBlocks_DMA()* and *HAL_SD_WriteBlocks_DMA()* APIs).
 - Enable the DMAx interface clock using *__HAL_RCC_DMAx_CLK_ENABLE()*;
 - Configure the DMA using the function *HAL_DMA_Init()* with predeclared and filled.
 - d. NVIC configuration if you need to use interrupt process when using DMA transfer.
 - Configure the SDIO and DMA interrupt priorities using functions *HAL_NVIC_SetPriority()*; DMA priority is superior to SDIO's priority

- Enable the NVIC DMA and SDIO IRQs using function `HAL_NVIC_EnableIRQ()`
 - SDIO interrupts are managed using the macros `__HAL_SD_SDIO_ENABLE_IT()` and `__HAL_SD_SDIO_DISABLE_IT()` inside the communication process.
 - SDIO interrupts pending bits are managed using the macros `__HAL_SD_SDIO_GET_IT()` and `__HAL_SD_SDIO_CLEAR_IT()`
2. At this stage, you can perform SD read/write/erase operations after SD card initialization

SD Card Initialization and configuration

To initialize the SD Card, use the `HAL_SD_Init()` function. It Initializes the SD Card and put it into Standby State (Ready for data transfer). This function provide the following operations:

1. Apply the SD Card initialization process at 400KHz and check the SD Card type (Standard Capacity or High Capacity). You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDIO_CK) is computed as follows: $SDIO_CK = SDIOCLK / (ClockDiv + 2)$ In initialization mode and according to the SD Card standard, make sure that the SDIO_CK frequency doesn't exceed 400KHz.
2. Get the SD CID and CSD data. All these information are managed by the `SDCardInfo` structure. This structure provide also ready computed SD Card capacity and Block size. These information are stored in SD handle structure in case of future use.
3. Configure the SD Card Data transfer frequency. By Default, the card transfer frequency is set to 48MHz / $(SDIO_TRANSFER_CLK_DIV + 2) = 8MHz$. You can change or adapt this frequency by adjusting the "ClockDiv" field. The SD Card frequency (SDIO_CK) is computed as follows: $SDIO_CK = SDIOCLK / (ClockDiv + 2)$ In transfer mode and according to the SD Card standard, make sure that the SDIO_CK frequency doesn't exceed 25MHz and 50MHz in High-speed mode switch. To be able to use a frequency higher than 24MHz, you should use the SDIO peripheral in bypass mode. Refer to the corresponding reference manual for more details.
4. Select the corresponding SD Card according to the address read with the step 2.
5. Configure the SD Card in wide bus mode: 4-bits data.

SD Card Read operation

- You can read from SD card in polling mode by using function `HAL_SD_ReadBlocks()`. This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.
- You can read from SD card in DMA mode by using function `HAL_SD_ReadBlocks_DMA()`. This function support only 512-bytes block length (the block size should be chosen as 512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to call the function `HAL_SD_CheckReadOperation()`, to insure that the read transfer is done correctly in both DMA and SD sides.

SD Card Write operation

- You can write to SD card in polling mode by using function `HAL_SD_WriteBlocks()`. This function support only 512-bytes block length (the block size should be chosen as

512 bytes). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter.

- You can write to SD card in DMA mode by using function `HAL_SD_WriteBlocks_DMA()`. This function support only 512-bytes block length (the block size should be chosen as 512 byte). You can choose either one block read operation or multiple block read operation by adjusting the "NumberOfBlocks" parameter. After this, you have to call the function `HAL_SD_CheckWriteOperation()`, to insure that the write transfer is done correctly in both DMA and SD sides.

SD card status

- At any time, you can check the SD Card status and get the SD card state by using the `HAL_SD_GetStatus()` function. This function checks first if the SD card is still connected and then get the internal SD Card transfer state.
- You can also get the SD card SD Status register by using the `HAL_SD_SendSDStatus()` function.

SD HAL driver macros list



You can refer to the SD HAL driver header file for more useful macros

37.3.2 Initialization and de-initialization functions

This section provides functions allowing to initialize/de-initialize the SD card device to be ready for use.

This section contains the following APIs:

- [*HAL_SD_Init\(\)*](#)
- [*HAL_SD_DeInit\(\)*](#)
- [*HAL_SD_MspInit\(\)*](#)
- [*HAL_SD_MspDeInit\(\)*](#)

37.3.3 IO operation functions

This subsection provides a set of functions allowing to manage the data transfer from/to SD card.

This section contains the following APIs:

- [*HAL_SD_ReadBlocks\(\)*](#)
- [*HAL_SD_WriteBlocks\(\)*](#)
- [*HAL_SD_ReadBlocks_DMA\(\)*](#)
- [*HAL_SD_WriteBlocks_DMA\(\)*](#)
- [*HAL_SD_CheckReadOperation\(\)*](#)
- [*HAL_SD_CheckWriteOperation\(\)*](#)
- [*HAL_SD_Erase\(\)*](#)
- [*HAL_SD_IRQHandler\(\)*](#)
- [*HAL_SD_XferCpltCallback\(\)*](#)
- [*HAL_SD_XferErrorCallback\(\)*](#)
- [*HAL_SD_DMA_RxCpltCallback\(\)*](#)
- [*HAL_SD_DMA_RxErrorCallback\(\)*](#)

- [HAL_SD_DMA_TxCpltCallback\(\)](#)
- [HAL_SD_DMA_TxErrorCallback\(\)](#)

37.3.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the SD card operations.

This section contains the following APIs:

- [HAL_SD_Get_CardInfo\(\)](#)
- [HAL_SD_WideBusOperation_Config\(\)](#)
- [HAL_SD_StopTransfer\(\)](#)
- [HAL_SD_HighSpeed\(\)](#)

37.3.5 Peripheral State functions

This subsection permits to get in runtime the status of the peripheral and the data flow.

This section contains the following APIs:

- [HAL_SD_SendSDStatus\(\)](#)
- [HAL_SD_GetStatus\(\)](#)
- [HAL_SD_GetCardStatus\(\)](#)

37.3.6 HAL_SD_Init

Function Name	HAL_SD_ErrorTypeDef HAL_SD_Init (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * SDCardInfo)
Function Description	Initializes the SD card according to the specified parameters in the SD_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • SDCardInfo: HAL_SD_CardInfoTypeDef structure for SD card information
Return values	<ul style="list-style-type: none"> • HAL SD error state

37.3.7 HAL_SD_DeInit

Function Name	HAL_StatusTypeDef HAL_SD_DeInit (SD_HandleTypeDef * hsd)
Function Description	De-Initializes the SD card.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • HAL status

37.3.8 HAL_SD_MspInit

Function Name	void HAL_SD_MspInit (SD_HandleTypeDef * hsd)
Function Description	Initializes the SD MSP.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • None

37.3.9 HAL_SD_MspDeInit

Function Name	void HAL_SD_MspDeInit (SD_HandleTypeDef * hsd)
Function Description	De-Initialize SD MSP.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • None

37.3.10 HAL_SD_ReadBlocks

Function Name	HAL_SD_ErrorTypeDef HAL_SD_ReadBlocks (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)
Function Description	Reads block(s) from a specified address in a card.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • pReadBuffer: pointer to the buffer that will contain the received data • ReadAddr: Address from where data is to be read • BlockSize: SD card Data block size • NumberOfBlocks: Number of SD blocks to read
Return values	<ul style="list-style-type: none"> • SD Card error state
Notes	<ul style="list-style-type: none"> • BlockSize must be 512 bytes.

37.3.11 HAL_SD_WriteBlocks

Function Name	HAL_SD_ErrorTypeDef HAL_SD_WriteBlocks (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)
Function Description	Allows to write block(s) to a specified address in a card.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • pWriteBuffer: pointer to the buffer that will contain the data to transmit • WriteAddr: Address from where data is to be written • BlockSize: SD card Data block size • NumberOfBlocks: Number of SD blocks to write
Return values	<ul style="list-style-type: none"> • SD Card error state
Notes	<ul style="list-style-type: none"> • BlockSize must be 512 bytes.

37.3.12 HAL_SD_ReadBlocks_DMA

Function Name	HAL_SD_ErrorTypeDef HAL_SD_ReadBlocks_DMA (SD_HandleTypeDef * hsd, uint32_t * pReadBuffer, uint64_t ReadAddr, uint32_t BlockSize, uint32_t NumberOfBlocks)
Function Description	Reads block(s) from a specified address in a card.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle • pReadBuffer: Pointer to the buffer that will contain the received data • ReadAddr: Address from where data is to be read • BlockSize: SD card Data block size • NumberOfBlocks: Number of blocks to read.

- | | |
|---------------|---|
| Return values | <ul style="list-style-type: none"> • SD Card error state |
| Notes | <ul style="list-style-type: none"> • This API should be followed by the function <code>HAL_SD_CheckReadOperation()</code> to check the completion of the read process • <code>BlockSize</code> must be 512 bytes. |

37.3.13 HAL_SD_WriteBlocks_DMA

- | | |
|----------------------|--|
| Function Name | HAL_SD_ErrorTypeDef HAL_SD_WriteBlocks_DMA (SD_HandleTypeDef * hsd, uint32_t * pWriteBuffer, uint64_t WriteAddr, uint32_t BlockSize, uint32_t NumberOfBlocks) |
| Function Description | Writes block(s) to a specified address in a card. |
| Parameters | <ul style="list-style-type: none"> • hsd: SD handle • pWriteBuffer: pointer to the buffer that will contain the data to transmit • WriteAddr: Address from where data is to be read • BlockSize: the SD card Data block size • NumberOfBlocks: Number of blocks to write |
| Return values | <ul style="list-style-type: none"> • SD Card error state |
| Notes | <ul style="list-style-type: none"> • This API should be followed by the function <code>HAL_SD_CheckWriteOperation()</code> to check the completion of the write process (by SD current status polling). • <code>BlockSize</code> must be 512 bytes. |

37.3.14 HAL_SD_CheckReadOperation

- | | |
|----------------------|---|
| Function Name | HAL_SD_ErrorTypeDef HAL_SD_CheckReadOperation (SD_HandleTypeDef * hsd, uint32_t Timeout) |
| Function Description | This function waits until the SD DMA data read transfer is finished. |
| Parameters | <ul style="list-style-type: none"> • hsd: SD handle • Timeout: Timeout duration |
| Return values | <ul style="list-style-type: none"> • SD Card error state |

37.3.15 HAL_SD_CheckWriteOperation

- | | |
|----------------------|---|
| Function Name | HAL_SD_ErrorTypeDef HAL_SD_CheckWriteOperation (SD_HandleTypeDef * hsd, uint32_t Timeout) |
| Function Description | This function waits until the SD DMA data write transfer is finished. |
| Parameters | <ul style="list-style-type: none"> • hsd: SD handle • Timeout: Timeout duration |
| Return values | <ul style="list-style-type: none"> • SD Card error state |

37.3.16 HAL_SD_Erase

- | | |
|----------------------|--|
| Function Name | HAL_SD_ErrorTypeDef HAL_SD_Erase (SD_HandleTypeDef * hsd, uint64_t startaddr, uint64_t endaddr) |
| Function Description | Erases the specified memory area of the given SD card. |

Parameters	<ul style="list-style-type: none"> • hsd: SD handle • startaddr: Start byte address • endaddr: End byte address
Return values	<ul style="list-style-type: none"> • SD Card error state

37.3.17 HAL_SD_IRQHandler

Function Name	void HAL_SD_IRQHandler (SD_HandleTypeDef * hsd)
Function Description	This function handles SD card interrupt request.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • None

37.3.18 HAL_SD_XferCpltCallback

Function Name	void HAL_SD_XferCpltCallback (SD_HandleTypeDef * hsd)
Function Description	SD end of transfer callback.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • None

37.3.19 HAL_SD_XferErrorCallback

Function Name	void HAL_SD_XferErrorCallback (SD_HandleTypeDef * hsd)
Function Description	SD Transfer Error callback.
Parameters	<ul style="list-style-type: none"> • hsd: SD handle
Return values	<ul style="list-style-type: none"> • None

37.3.20 HAL_SD_DMA_RxCpltCallback

Function Name	void HAL_SD_DMA_RxCpltCallback (DMA_HandleTypeDef * hdma)
Function Description	SD Transfer complete Rx callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	<ul style="list-style-type: none"> • None

37.3.21 HAL_SD_DMA_RxErrorCallback

Function Name	void HAL_SD_DMA_RxErrorCallback (DMA_HandleTypeDef * hdma)
Function Description	SD DMA transfer complete Rx error callback.
Parameters	<ul style="list-style-type: none"> • hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	<ul style="list-style-type: none"> • None

37.3.22 HAL_SD_DMA_TxCpltCallback

Function Name	void HAL_SD_DMA_TxCpltCallback (DMA_HandleTypeDef * hdma)
Function Description	SD Transfer complete Tx callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	<ul style="list-style-type: none"> None

37.3.23 HAL_SD_DMA_TxErrorCallback

Function Name	void HAL_SD_DMA_TxErrorCallback (DMA_HandleTypeDef * hdma)
Function Description	SD DMA transfer complete error Tx callback.
Parameters	<ul style="list-style-type: none"> hdma: pointer to a DMA_HandleTypeDef structure that contains the configuration information for the specified DMA module.
Return values	<ul style="list-style-type: none"> None

37.3.24 HAL_SD_Get_CardInfo

Function Name	HAL_SD_ErrorTypeDef HAL_SD_Get_CardInfo (SD_HandleTypeDef * hsd, HAL_SD_CardInfoTypeDef * pCardInfo)
Function Description	Returns information about specific card.
Parameters	<ul style="list-style-type: none"> hsd: SD handle pCardInfo: Pointer to a HAL_SD_CardInfoTypeDef structure that contains all SD card information
Return values	<ul style="list-style-type: none"> SD Card error state

37.3.25 HAL_SD_WideBusOperation_Config

Function Name	HAL_SD_ErrorTypeDef HAL_SD_WideBusOperation_Config (SD_HandleTypeDef * hsd, uint32_t WideMode)
Function Description	Enables wide bus operation for the requested card if supported by card.
Parameters	<ul style="list-style-type: none"> hsd: SD handle WideMode: Specifies the SD card wide bus mode This parameter can be one of the following values: SDIO_BUS_WIDE_8B: 8-bit data transfer (Only for MMC)SDIO_BUS_WIDE_4B: 4-bit data transferSDIO_BUS_WIDE_1B: 1-bit data transfer
Return values	<ul style="list-style-type: none"> SD Card error state

37.3.26 HAL_SD_StopTransfer

Function Name	HAL_SD_ErrorTypeDef HAL_SD_StopTransfer
---------------	--

(SD_HandleTypeDef * hsd)

Function Description Aborts an ongoing data transfer.

Parameters

- **hsd:** SD handle

Return values

- SD Card error state

37.3.27 HAL_SD_HighSpeed

Function Name **HAL_SD_ErrorTypeDef HAL_SD_HighSpeed**
(SD_HandleTypeDef * hsd)

Function Description Switches the SD card to High Speed mode.

Parameters

- **hsd:** SD handle

Return values

- SD Card error state

Notes

- This operation should be followed by the configuration of PLL to have SDIOCK clock between 67 and 75 MHz

37.3.28 HAL_SD_SendSDStatus

Function Name **HAL_SD_ErrorTypeDef HAL_SD_SendSDStatus**
(SD_HandleTypeDef * hsd, uint32_t * pSDstatus)

Function Description Returns the current SD card's status.

Parameters

- **hsd:** SD handle
- **pSDstatus:** Pointer to the buffer that will contain the SD card status SD Status register)

Return values

- SD Card error state

37.3.29 HAL_SD_GetStatus

Function Name **HAL_SD_TransferStateTypeDef HAL_SD_GetStatus**
(SD_HandleTypeDef * hsd)

Function Description Gets the current sd card data status.

Parameters

- **hsd:** SD handle

Return values

- Data Transfer state

37.3.30 HAL_SD_GetCardStatus

Function Name **HAL_SD_ErrorTypeDef HAL_SD_GetCardStatus**
(SD_HandleTypeDef * hsd, HAL_SD_CardStatusTypeDef * pCardStatus)

Function Description Gets the SD card status.

Parameters

- **hsd:** SD handle
- **pCardStatus:** Pointer to the HAL_SD_CardStatusTypeDef structure that will contain the SD card status information

Return values

- SD Card error state

37.4 SD Firmware driver defines

37.4.1 SD

SD Exported Constants

SD_CMD_GO_IDLE_STATE	Resets the SD memory card.
SD_CMD_SEND_OP_COND	Sends host capacity support information and activates the card's initialization process.
SD_CMD_ALL_SEND_CID	Asks any card connected to the host to send the CID numbers on the CMD line.
SD_CMD_SET_REL_ADDR	Asks the card to publish a new relative address (RCA).
SD_CMD_SET_DSR	Programs the DSR of all cards.
SD_CMD_SDIO_SEN_OP_COND	Sends host capacity support information (HCS) and asks the accessed card to send its operating condition register (OCR) content in the response on the CMD line.
SD_CMD_HS_SWITCH	Checks switchable function (mode 0) and switch card function (mode 1).
SD_CMD_SEL_DESEL_CARD	Selects the card by its own relative address and gets deselected by any other address
SD_CMD_HS_SEND_EXT_CSD	Sends SD Memory Card interface condition, which includes host supply voltage information and asks the card whether card supports voltage.
SD_CMD_SEND_CSD	Addressed card sends its card specific data (CSD) on the CMD line.
SD_CMD_SEND_CID	Addressed card sends its card identification (CID) on the CMD line.
SD_CMD_READ_DAT_UNTIL_STOP	SD card doesn't support it.
SD_CMD_STOP_TRANSMISSION	Forces the card to stop transmission.
SD_CMD_SEND_STATUS	Addressed card sends its

	status register.
SD_CMD_HS_BUSTEST_READ	
SD_CMD_GO_INACTIVE_STATE	Sends an addressed card into the inactive state.
SD_CMD_SET_BLOCKLEN	Sets the block length (in bytes for SDSC) for all following block commands (read, write, lock). Default block length is fixed to 512 Bytes. Not effective for SDHS and SDXC.
SD_CMD_READ_SINGLE_BLOCK	Reads single block of size selected by SET_BLOCKLEN in case of SDSC, and a block of fixed 512 bytes in case of SDHC and SDXC.
SD_CMD_READ_MULT_BLOCK	Continuously transfers data blocks from card to host until interrupted by STOP_TRANSMISSION command.
SD_CMD_HS_BUSTEST_WRITE	64 bytes tuning pattern is sent for SDR50 and SDR104.
SD_CMD_WRITE_DAT_UNTIL_STOP	Speed class control command.
SD_CMD_SET_BLOCK_COUNT	Specify block count for CMD18 and CMD25.
SD_CMD_WRITE_SINGLE_BLOCK	Writes single block of size selected by SET_BLOCKLEN in case of SDSC, and a block of fixed 512 bytes in case of SDHC and SDXC.
SD_CMD_WRITE_MULT_BLOCK	Continuously writes blocks of data until a STOP_TRANSMISSION follows.
SD_CMD_PROG_CID	Reserved for manufacturers.
SD_CMD_PROG_CSD	Programming of the programmable bits of the CSD.
SD_CMD_SET_WRITE_PROT	Sets the write protection bit of the addressed group.
SD_CMD_CLR_WRITE_PROT	Clears the write protection bit of the addressed group.

SD_CMD_SEND_WRITE_PROT	Asks the card to send the status of the write protection bits.
SD_CMD_SD_ERASE_GRP_START	Sets the address of the first write block to be erased. (For SD card only).
SD_CMD_SD_ERASE_GRP_END	Sets the address of the last write block of the continuous range to be erased.
SD_CMD_ERASE_GRP_START	Sets the address of the first write block to be erased. Reserved for each command system set by switch function command (CMD6).
SD_CMD_ERASE_GRP_END	Sets the address of the last write block of the continuous range to be erased. Reserved for each command system set by switch function command (CMD6).
SD_CMD_ERASE	Reserved for SD security applications.
SD_CMD_FAST_IO	SD card doesn't support it (Reserved).
SD_CMD_GO_IRQ_STATE	SD card doesn't support it (Reserved).
SD_CMD_LOCK_UNLOCK	Sets/resets the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command.
SD_CMD_APP_CMD	Indicates to the card that the next command is an application specific command rather than a standard command.
SD_CMD_GEN_CMD	Used either to transfer a data block to the card or to get a data block from the card for general purpose/application specific commands.
SD_CMD_NO_CMD	
SD_CMD_APP_SD_SET_BUSWIDTH	SDIO_APP_CMD should be sent before sending these commands. (ACMD6) Defines the data bus width

	to be used for data transfer. The allowed data bus widths are given in SCR register.
SD_CMD_SD_APP_STATUS	(ACMD13) Sends the SD status.
SD_CMD_SD_APP_SEND_NUM_WRITE_BLOCKS	(ACMD22) Sends the number of the written (without errors) write blocks. Responds with 32bit+CRC data block.
SD_CMD_SD_APP_OP_COND	(ACMD41) Sends host capacity support information (HCS) and asks the accessed card to send its operating condition register (OCR) content in the response on the CMD line.
SD_CMD_SD_APP_SET_CLR_CARD_DETECT	(ACMD42) Connects/Disconnects the 50 KOhm pull-up resistor on CD/DAT3 (pin 1) of the card.
SD_CMD_SD_APP_SEND_SCR	Reads the SD Configuration Register (SCR).
SD_CMD_SDIO_RW_DIRECT	For SD I/O card only, reserved for security specification.
SD_CMD_SDIO_RW_EXTENDED	For SD I/O card only, reserved for security specification.
SD_CMD_SD_APP_GET_MKB	SD_CMD_APP_CMD should be sent before sending these commands. For SD card only
SD_CMD_SD_APP_GET_MID	For SD card only
SD_CMD_SD_APP_SET_CER_RN1	For SD card only
SD_CMD_SD_APP_GET_CER_RN2	For SD card only
SD_CMD_SD_APP_SET_CER_RES2	For SD card only
SD_CMD_SD_APP_GET_CER_RES1	For SD card only
SD_CMD_SD_APP_SECURE_READ_MULTIPLE_BLOCK	For SD card only
SD_CMD_SD_APP_SECURE_WRITE_MULTIPLE_BLOCK	For SD card only
SD_CMD_SD_APP_SECURE_ERASE	For SD card only
SD_CMD_SD_APP_CHANGE_SECURE_AREA	For SD card only
SD_CMD_SD_APP_SECURE_WRITE_MKB	For SD card only
STD_CAPACITY_SD_CARD_V1_1	
STD_CAPACITY_SD_CARD_V2_0	

HIGH_CAPACITY_SD_CARD
MULTIMEDIA_CARD
SECURE_DIGITAL_IO_CARD
HIGH_SPEED_MULTIMEDIA_CARD
SECURE_DIGITAL_IO_COMBO_CARD
HIGH_CAPACITY_MMC_CARD

SD Exported Macros

__HAL_SD_SDIO_ENABLE

Description:

- Enable the SD device.

Return value:

- None

__HAL_SD_SDIO_DISABLE

Description:

- Disable the SD device.

Return value:

- None

__HAL_SD_SDIO_DMA_ENABLE

Description:

- Enable the SDIO DMA transfer.

Return value:

- None

__HAL_SD_SDIO_DMA_DISABLE

Description:

- Disable the SDIO DMA transfer.

Return value:

- None

__HAL_SD_SDIO_ENABLE_IT

Description:

- Enable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle
- __INTERRUPT__: specifies the SDIO interrupt sources to be enabled. This parameter can be one or a combination of the following values:
 - SDIO_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
 - SDIO_IT_DCRCFAIL: Data block sent/received (CRC check failed) interrupt
 - SDIO_IT_CTIMEOUT: Command response timeout interrupt
 - SDIO_IT_DTIMEOUT: Data timeout interrupt
 - SDIO_IT_TXUNDERR: Transmit FIFO underrun error interrupt
 - SDIO_IT_RXOVERR: Received FIFO

- overrun error interrupt
- SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
- SDIO_IT_CMDSSENT: Command sent (no response required) interrupt
- SDIO_IT_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO_IT_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO_IT_CMDACT: Command transfer in progress interrupt
- SDIO_IT_TXACT: Data transmit in progress interrupt
- SDIO_IT_RXACT: Data receive in progress interrupt
- SDIO_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO_IT_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO_IT_TXFIFO: Transmit FIFO full interrupt
- SDIO_IT_RXFIFO: Receive FIFO full interrupt
- SDIO_IT_TXFIFOE: Transmit FIFO empty interrupt
- SDIO_IT_RXFIFOE: Receive FIFO empty interrupt
- SDIO_IT_TXDAVL: Data available in transmit FIFO interrupt
- SDIO_IT_RXDAVL: Data available in receive FIFO interrupt
- SDIO_IT_SDIOIT: SD I/O interrupt received interrupt
- SDIO_IT_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

Return value:

- None

__HAL_SD_SDIO_DISABLE_IT**Description:**

- Disable the SD device interrupt.

Parameters:

- __HANDLE__: SD Handle
- __INTERRUPT__: specifies the SDIO interrupt sources to be disabled. This parameter can be one or a combination of the following values:
 - SDIO_IT_CCRCFAIL: Command response received (CRC check failed) interrupt
 - SDIO_IT_DCRCFAIL: Data block

- sent/received (CRC check failed) interrupt
- SDIO_IT_CTIMEOUT: Command response timeout interrupt
- SDIO_IT_DTIMEOUT: Data timeout interrupt
- SDIO_IT_TXUNDERR: Transmit FIFO underrun error interrupt
- SDIO_IT_RXOVERR: Received FIFO overrun error interrupt
- SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
- SDIO_IT_CMDSSENT: Command sent (no response required) interrupt
- SDIO_IT_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO_IT_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO_IT_CMDACT: Command transfer in progress interrupt
- SDIO_IT_TXACT: Data transmit in progress interrupt
- SDIO_IT_RXACT: Data receive in progress interrupt
- SDIO_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO_IT_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO_IT_TXFIFOOF: Transmit FIFO full interrupt
- SDIO_IT_RXFIFOOF: Receive FIFO full interrupt
- SDIO_IT_TXFIFOE: Transmit FIFO empty interrupt
- SDIO_IT_RXFIFOE: Receive FIFO empty interrupt
- SDIO_IT_TXDAVL: Data available in transmit FIFO interrupt
- SDIO_IT_RXDAVL: Data available in receive FIFO interrupt
- SDIO_IT_SDIOIT: SD I/O interrupt received interrupt
- SDIO_IT_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

Return value:

- None

__HAL_SD_SDIO_GET_FLAG**Description:**

- Check whether the specified SD flag is set or not.

Parameters:

- `__HANDLE__`: SD Handle
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `SDIO_FLAG_CCRCFAIL`: Command response received (CRC check failed)
 - `SDIO_FLAG_DCRCFAIL`: Data block sent/received (CRC check failed)
 - `SDIO_FLAG_CTIMEOUT`: Command response timeout
 - `SDIO_FLAG_DTIMEOUT`: Data timeout
 - `SDIO_FLAG_TXUNDERR`: Transmit FIFO underrun error
 - `SDIO_FLAG_RXOVERR`: Received FIFO overrun error
 - `SDIO_FLAG_CMDREND`: Command response received (CRC check passed)
 - `SDIO_FLAG_CMDSENT`: Command sent (no response required)
 - `SDIO_FLAG_DATAEND`: Data end (data counter, `SDIDCOUNT`, is zero)
 - `SDIO_FLAG_STBITERR`: Start bit not detected on all data signals in wide bus mode.
 - `SDIO_FLAG_DBCKEND`: Data block sent/received (CRC check passed)
 - `SDIO_FLAG_CMDACT`: Command transfer in progress
 - `SDIO_FLAG_TXACT`: Data transmit in progress
 - `SDIO_FLAG_RXACT`: Data receive in progress
 - `SDIO_FLAG_TXFIFOHE`: Transmit FIFO Half Empty
 - `SDIO_FLAG_RXFIFOHF`: Receive FIFO Half Full
 - `SDIO_FLAG_TXFIFO`: Transmit FIFO full
 - `SDIO_FLAG_RXFIFO`: Receive FIFO full
 - `SDIO_FLAG_TXFIFOE`: Transmit FIFO empty
 - `SDIO_FLAG_RXFIFOE`: Receive FIFO empty
 - `SDIO_FLAG_TXDAVL`: Data available in transmit FIFO
 - `SDIO_FLAG_RXDAVL`: Data available in receive FIFO
 - `SDIO_FLAG_SDIOIT`: SD I/O interrupt received
 - `SDIO_FLAG_CEATAEND`: CE-ATA command completion signal received for CMD61

Return value:

`__HAL_SD_SDIO_CLEAR_FLAG`

- The: new state of SD FLAG (SET or RESET).

Description:

- Clear the SD's pending flags.

Parameters:

- `__HANDLE__`: SD Handle
- `__FLAG__`: specifies the flag to clear. This parameter can be one or a combination of the following values:
 - `SDIO_FLAG_CCRCFAIL`: Command response received (CRC check failed)
 - `SDIO_FLAG_DCRCFAIL`: Data block sent/received (CRC check failed)
 - `SDIO_FLAG_CTIMEOUT`: Command response timeout
 - `SDIO_FLAG_DTIMEOUT`: Data timeout
 - `SDIO_FLAG_TXUNDERR`: Transmit FIFO underrun error
 - `SDIO_FLAG_RXOVERR`: Received FIFO overrun error
 - `SDIO_FLAG_CMDREND`: Command response received (CRC check passed)
 - `SDIO_FLAG_CMDSSENT`: Command sent (no response required)
 - `SDIO_FLAG_DATAEND`: Data end (data counter, `SDIDCOUNT`, is zero)
 - `SDIO_FLAG_STBITERR`: Start bit not detected on all data signals in wide bus mode
 - `SDIO_FLAG_DBCKEND`: Data block sent/received (CRC check passed)
 - `SDIO_FLAG_SDIOIT`: SD I/O interrupt received
 - `SDIO_FLAG_CEATAEND`: CE-ATA command completion signal received for CMD61

Return value:

- None

`__HAL_SD_SDIO_GET_IT`**Description:**

- Check whether the specified SD interrupt has occurred or not.

Parameters:

- `__HANDLE__`: SD Handle
- `__INTERRUPT__`: specifies the SDIO interrupt source to check. This parameter can be one of the following values:
 - `SDIO_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
 - `SDIO_IT_DCRCFAIL`: Data block sent/received (CRC check failed) interrupt

- SDIO_IT_CTIMEOUT: Command response timeout interrupt
- SDIO_IT_DTIMEOUT: Data timeout interrupt
- SDIO_IT_TXUNDERR: Transmit FIFO underrun error interrupt
- SDIO_IT_RXOVERR: Received FIFO overrun error interrupt
- SDIO_IT_CMDREND: Command response received (CRC check passed) interrupt
- SDIO_IT_CMDSSENT: Command sent (no response required) interrupt
- SDIO_IT_DATAEND: Data end (data counter, SDIDCOUNT, is zero) interrupt
- SDIO_IT_STBITERR: Start bit not detected on all data signals in wide bus mode interrupt
- SDIO_IT_DBCKEND: Data block sent/received (CRC check passed) interrupt
- SDIO_IT_CMDACT: Command transfer in progress interrupt
- SDIO_IT_TXACT: Data transmit in progress interrupt
- SDIO_IT_RXACT: Data receive in progress interrupt
- SDIO_IT_TXFIFOHE: Transmit FIFO Half Empty interrupt
- SDIO_IT_RXFIFOHF: Receive FIFO Half Full interrupt
- SDIO_IT_TXFIFOOF: Transmit FIFO full interrupt
- SDIO_IT_RXFIFOOF: Receive FIFO full interrupt
- SDIO_IT_TXFIFOE: Transmit FIFO empty interrupt
- SDIO_IT_RXFIFOE: Receive FIFO empty interrupt
- SDIO_IT_TXDAVL: Data available in transmit FIFO interrupt
- SDIO_IT_RXDAVL: Data available in receive FIFO interrupt
- SDIO_IT_SDIOIT: SD I/O interrupt received interrupt
- SDIO_IT_CEATAEND: CE-ATA command completion signal received for CMD61 interrupt

Return value:

- The: new state of SD IT (SET or RESET).

Description:

- Clear the SD's interrupt pending bits.

Parameters:

`__HAL_SD_SDIO_CLEAR_IT`

- `__HANDLE__`: : SD Handle
- `__INTERRUPT__`: specifies the interrupt pending bit to clear. This parameter can be one or a combination of the following values:
 - `SDIO_IT_CCRCFAIL`: Command response received (CRC check failed) interrupt
 - `SDIO_IT_DCRCFAIL`: Data block sent/received (CRC check failed) interrupt
 - `SDIO_IT_CTIMEOUT`: Command response timeout interrupt
 - `SDIO_IT_DTIMEOUT`: Data timeout interrupt
 - `SDIO_IT_TXUNDERR`: Transmit FIFO underrun error interrupt
 - `SDIO_IT_RXOVERR`: Received FIFO overrun error interrupt
 - `SDIO_IT_CMDREND`: Command response received (CRC check passed) interrupt
 - `SDIO_IT_CMDSSENT`: Command sent (no response required) interrupt
 - `SDIO_IT_DATAEND`: Data end (data counter, `SDIO_DCOUNT`, is zero) interrupt
 - `SDIO_IT_STBITERR`: Start bit not detected on all data signals in wide bus mode interrupt
 - `SDIO_IT_SDIOIT`: SD I/O interrupt received interrupt
 - `SDIO_IT_CEATAEND`: CE-ATA command completion signal received for CMD61

Return value:

- None

SD Handle Structure definition`SD_InitTypeDef``SD_TypeDef`***SD Private Defines***`DATA_BLOCK_SIZE``SDIO_STATIC_FLAGS``SDIO_CMD0TIMEOUT``SD_OCR_ADDR_OUT_OF_RANGE``SD_OCR_ADDR_MISALIGNED``SD_OCR_BLOCK_LEN_ERR``SD_OCR_ERASE_SEQ_ERR``SD_OCR_BAD_ERASE_PARAM``SD_OCR_WRITE_PROT_VIOLATION``SD_OCR_LOCK_UNLOCK_FAILED`

SD_OCR_COM_CRC_FAILED
SD_OCR_ILLEGAL_CMD
SD_OCR_CARD_ECC_FAILED
SD_OCR_CC_ERROR
SD_OCR_GENERAL_UNKNOWN_ERROR
SD_OCR_STREAM_READ_UNDERRUN
SD_OCR_STREAM_WRITE_OVERRUN
SD_OCR_CID_CSD_OVERWRITE
SD_OCR_WP_ERASE_SKIP
SD_OCR_CARD_ECC_DISABLED
SD_OCR_ERASE_RESET
SD_OCR_AKE_SEQ_ERROR
SD_OCR_ERRORBITS
SD_R6_GENERAL_UNKNOWN_ERROR
SD_R6_ILLEGAL_CMD
SD_R6_COM_CRC_FAILED
SD_VOLTAGE_WINDOW_SD
SD_HIGH_CAPACITY
SD_STD_CAPACITY
SD_CHECK_PATTERN
SD_MAX_VOLT_TRIAL
SD_ALLZERO
SD_WIDE_BUS_SUPPORT
SD_SINGLE_BUS_SUPPORT
SD_CARD_LOCKED
SD_DATATIMEOUT
SD_0TO7BITS
SD_8TO15BITS
SD_16TO23BITS
SD_24TO31BITS
SD_MAX_DATA_LENGTH
SD_HALFFIFO
SD_HALFFIFOBYTES
SD_CCCC_LOCK_UNLOCK
SD_CCCC_WRITE_PROT
SD_CCCC_ERASE

`SD_SDIO_SEND_IF_COND`

`SDIO_APP_CMD` should be sent before sending these commands.

38 HAL SMARTCARD Generic Driver

38.1 HAL SMARTCARD Generic Driver

38.2 SMARTCARD Firmware driver registers structures

38.2.1 SMARTCARD_InitTypeDef

Data Fields

- *uint32_t BaudRate*
- *uint32_t WordLength*
- *uint32_t StopBits*
- *uint32_t Parity*
- *uint32_t Mode*
- *uint32_t CLKPolarity*
- *uint32_t CLKPhase*
- *uint32_t CLKLastBit*
- *uint32_t Prescaler*
- *uint32_t GuardTime*
- *uint32_t NACKState*

Field Documentation

- ***uint32_t SMARTCARD_InitTypeDef::BaudRate***
This member configures the SmartCard communication baud rate. The baud rate is computed using the following formula:
$$\text{IntegerDivider} = ((\text{PCLKx}) / (16 * (\text{hsmartcard->Init.BaudRate})))$$
$$\text{FractionalDivider} = ((\text{IntegerDivider} - ((\text{uint32_t}) \text{IntegerDivider})) * 16) + 0.5$$
- ***uint32_t SMARTCARD_InitTypeDef::WordLength***
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [SMARTCARD_Word_Length](#)
- ***uint32_t SMARTCARD_InitTypeDef::StopBits***
Specifies the number of stop bits transmitted. This parameter can be a value of [SMARTCARD_Stop_Bits](#)
- ***uint32_t SMARTCARD_InitTypeDef::Parity***
Specifies the parity mode. This parameter can be a value of [SMARTCARD_Parity](#)
Note:When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t SMARTCARD_InitTypeDef::Mode***
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [SMARTCARD_Mode](#)
- ***uint32_t SMARTCARD_InitTypeDef::CLKPolarity***
Specifies the steady state of the serial clock. This parameter can be a value of [SMARTCARD_Clock_Polarity](#)
- ***uint32_t SMARTCARD_InitTypeDef::CLKPhase***
Specifies the clock transition on which the bit capture is made. This parameter can be a value of [SMARTCARD_Clock_Phase](#)

- ***uint32_t SMARTCARD_InitTypeDef::CLKLastBit***
Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of [SMARTCARD_Last_Bit](#)
- ***uint32_t SMARTCARD_InitTypeDef::Prescaler***
Specifies the SmartCard Prescaler value used for dividing the system clock to provide the smartcard clock This parameter can be a value of [SMARTCARD_Prescaler](#)
- ***uint32_t SMARTCARD_InitTypeDef::GuardTime***
Specifies the SmartCard Guard Time value in terms of number of baud clocks The value given in the register (5 significant bits) is multiplied by 2 to give the division factor of the source clock frequency
- ***uint32_t SMARTCARD_InitTypeDef::NACKState***
Specifies the SmartCard NACK Transmission state This parameter can be a value of [SMARTCARD_NACK_State](#)

38.2.2 SMARTCARD_HandleTypeDef

Data Fields

- ***USART_TypeDef * Instance***
- ***SMARTCARD_InitTypeDef Init***
- ***uint8_t * pTxBuffPtr***
- ***uint16_t TxXferSize***
- ***uint16_t TxXferCount***
- ***uint8_t * pRxBuffPtr***
- ***uint16_t RxXferSize***
- ***uint16_t RxXferCount***
- ***DMA_HandleTypeDef * hdmatx***
- ***DMA_HandleTypeDef * hdmarx***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_SMARTCARD_StateTypeDef State***
- ***__IO uint32_t ErrorCode***

Field Documentation

- ***USART_TypeDef* SMARTCARD_HandleTypeDef::Instance***
USART registers base address
- ***SMARTCARD_InitTypeDef SMARTCARD_HandleTypeDef::Init***
SmartCard communication parameters
- ***uint8_t* SMARTCARD_HandleTypeDef::pTxBuffPtr***
Pointer to SmartCard Tx transfer Buffer
- ***uint16_t SMARTCARD_HandleTypeDef::TxXferSize***
SmartCard Tx Transfer size
- ***uint16_t SMARTCARD_HandleTypeDef::TxXferCount***
SmartCard Tx Transfer Counter
- ***uint8_t* SMARTCARD_HandleTypeDef::pRxBuffPtr***
Pointer to SmartCard Rx transfer Buffer
- ***uint16_t SMARTCARD_HandleTypeDef::RxXferSize***
SmartCard Rx Transfer size
- ***uint16_t SMARTCARD_HandleTypeDef::RxXferCount***
SmartCard Rx Transfer Counter

- ***DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmatx***
SmartCard Tx DMA Handle parameters
- ***DMA_HandleTypeDef* SMARTCARD_HandleTypeDef::hdmarx***
SmartCard Rx DMA Handle parameters
- ***HAL_LockTypeDef SMARTCARD_HandleTypeDef::Lock***
Locking object
- ***__IO HAL_SMARTCARD_StateTypeDef SMARTCARD_HandleTypeDef::State***
SmartCard communication state
- ***__IO uint32_t SMARTCARD_HandleTypeDef::ErrorCode***
SmartCard Error code

38.3 SMARTCARD Firmware driver API description

38.3.1 How to use this driver

The SMARTCARD HAL driver can be used as follows:

1. Declare a SMARTCARD_HandleTypeDef handle structure.
2. Initialize the SMARTCARD low level resources by implementing the HAL_SMARTCARD_MspInit() API:
 - a. Enable the interface clock of the USARTx associated to the SMARTCARD.
 - b. SMARTCARD pins configuration:
 - Enable the clock for the SMARTCARD GPIOs.
 - Configure the SMARTCARD pins as alternate function pull-up.
 - c. NVIC configuration if you need to use interrupt process (HAL_SMARTCARD_Transmit_IT() and HAL_SMARTCARD_Receive_IT() APIs):
 - Configure the USARTx interrupt priority.
 - Enable the NVIC USART IRQ handle.
 - d. DMA Configuration if you need to use DMA process (HAL_SMARTCARD_Transmit_DMA() and HAL_SMARTCARD_Receive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx channel.
 - Associate the initialized DMA handle to the SMARTCARD DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
 - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the SMARTCARD_Init structure.
4. Initialize the SMARTCARD registers by calling the HAL_SMARTCARD_Init() API:
 - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SMARTCARD_MspInit(&hsc) API. The specific SMARTCARD interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_SMARTCARD_ENABLE_IT() and __HAL_SMARTCARD_DISABLE_IT() inside the transmit and receive process.
5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_SMARTCARD_Transmit()
- Receive an amount of data in blocking mode using HAL_SMARTCARD_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_SMARTCARD_Transmit_IT()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_SMARTCARD_Receive_IT()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_SMARTCARD_Transmit_DMA()
- At transmission end of transfer HAL_SMARTCARD_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_SMARTCARD_Receive_DMA()
- At reception end of transfer HAL_SMARTCARD_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_RxCpltCallback
- In case of transfer Error, HAL_SMARTCARD_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_SMARTCARD_ErrorCallback

SMARTCARD HAL driver macros list

Below the list of most used macros in SMARTCARD HAL driver.

- __HAL_SMARTCARD_ENABLE: Enable the SMARTCARD peripheral
- __HAL_SMARTCARD_DISABLE: Disable the SMARTCARD peripheral
- __HAL_SMARTCARD_GET_FLAG : Check whether the specified SMARTCARD flag is set or not
- __HAL_SMARTCARD_CLEAR_FLAG : Clear the specified SMARTCARD pending flag
- __HAL_SMARTCARD_ENABLE_IT: Enable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_DISABLE_IT: Disable the specified SMARTCARD interrupt
- __HAL_SMARTCARD_GET_IT_SOURCE: Check whether the specified SMARTCARD interrupt has occurred or not



You can refer to the SMARTCARD HAL driver header file for more useful macros

38.3.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in Smartcard mode.

The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard.

The USART can provide a clock to the smartcard through the SCLK output. In smartcard mode, SCLK is not associated to the communication but is simply derived from the internal peripheral input clock through a 5-bit prescaler.

- For the Smartcard mode only these parameters can be configured:
 - Baud Rate
 - Word Length => Should be 9 bits (8 bits + parity)
 - Stop Bit
 - Parity: => Should be enabled (see [Table 23: "Smartcard frame formats"](#))
 - USART polarity
 - USART phase
 - USART LastBit
 - Receiver/transmitter modes
 - Prescaler
 - GuardTime
 - NACKState: The Smartcard NACK state
- Recommended SmartCard interface configuration to get the Answer to Reset from the Card:
 - Word Length = 9 Bits
 - 1.5 Stop Bit
 - Even parity
 - BaudRate = 12096 baud
 - Tx and Rx enabled

Table 23: Smartcard frame formats

M bit	PCE bit	Smartcard frame
1	1	SB 8 bit data PB STB



It is also possible to choose 0.5 stop bit for receiving but it is recommended to use 1.5 stop bits for both transmitting and receiving to avoid switching between the two configurations.

The HAL_SMARTCARD_Init() function follows the USART SmartCard configuration procedure (details for the procedure are available in reference manual (RM0038)).

This section contains the following APIs:

- [HAL_SMARTCARD_Init\(\)](#)
- [HAL_SMARTCARD_DeInit\(\)](#)
- [HAL_SMARTCARD_MspInit\(\)](#)

- [*HAL_SMARTCARD_MspDeInit\(\)*](#)

38.3.3 IO operation functions

This subsection provides a set of functions allowing to manage the SMARTCARD data transfers.

1. Smartcard is a single wire half duplex communication protocol. The Smartcard interface is designed to support asynchronous protocol Smartcards as defined in the ISO 7816-3 standard.
2. The USART should be configured as:
 - 8 bits plus parity: where M=1 and PCE=1 in the USART_CR1 register
 - 1.5 stop bits when transmitting and receiving: where STOP=11 in the USART_CR2 register.
3. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA, the relevant API's return the HAL status. The end of the data processing will be indicated through the dedicated SMARTCARD IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SMARTCARD_TxCpltCallback(), HAL_SMARTCARD_RxCpltCallback() user callbacks will be executed respectively at the end of the Transmit or Receive process. The HAL_SMARTCARD_ErrorCallback() user callback will be executed when a communication error is detected.
4. Blocking mode APIs are :
 - HAL_SMARTCARD_Transmit()
 - HAL_SMARTCARD_Receive()
5. Non Blocking mode APIs with Interrupt are :
 - HAL_SMARTCARD_Transmit_IT()
 - HAL_SMARTCARD_Receive_IT()
 - HAL_SMARTCARD_IRQHandler()
6. Non Blocking mode functions with DMA are :
 - HAL_SMARTCARD_Transmit_DMA()
 - HAL_SMARTCARD_Receive_DMA()
7. A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - HAL_SMARTCARD_TxCpltCallback()
 - HAL_SMARTCARD_RxCpltCallback()
 - HAL_SMARTCARD_ErrorCallback()

This section contains the following APIs:

- [*HAL_SMARTCARD_Transmit\(\)*](#)
- [*HAL_SMARTCARD_Receive\(\)*](#)
- [*HAL_SMARTCARD_Transmit_IT\(\)*](#)
- [*HAL_SMARTCARD_Receive_IT\(\)*](#)
- [*HAL_SMARTCARD_Transmit_DMA\(\)*](#)
- [*HAL_SMARTCARD_Receive_DMA\(\)*](#)
- [*HAL_SMARTCARD_IRQHandler\(\)*](#)
- [*HAL_SMARTCARD_TxCpltCallback\(\)*](#)
- [*HAL_SMARTCARD_RxCpltCallback\(\)*](#)
- [*HAL_SMARTCARD_ErrorCallback\(\)*](#)

38.3.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of SmartCard communication process and also return Peripheral Errors occurred during communication process

- HAL_SMARTCARD_GetState() API can be helpful to check in run-time the state of the SMARTCARD peripheral.
- HAL_SMARTCARD_GetError() check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- [HAL_SMARTCARD_GetState\(\)](#)
- [HAL_SMARTCARD_GetError\(\)](#)

38.3.5 HAL_SMARTCARD_Init

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Init (SMARTCARD_HandleTypeDef * hsc)
Function Description	Initializes the SmartCard mode according to the specified parameters in the SMARTCARD_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> • HAL status

38.3.6 HAL_SMARTCARD_DeInit

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_DeInit (SMARTCARD_HandleTypeDef * hsc)
Function Description	DeInitializes the SMARTCARD peripheral.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> • HAL status

38.3.7 HAL_SMARTCARD_Msplnit

Function Name	void HAL_SMARTCARD_Msplnit (SMARTCARD_HandleTypeDef * hsc)
Function Description	SMARTCARD MSP Init.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> • None

38.3.8 HAL_SMARTCARD_MspDeInit

Function Name	void HAL_SMARTCARD_MspDeInit (SMARTCARD_HandleTypeDef * hsc)
---------------	---

Function Description	SMARTCARD MSP DeInit.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> • None

38.3.9 HAL_SMARTCARD_Transmit

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Sends an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Specify timeout value
Return values	<ul style="list-style-type: none"> • HAL status

38.3.10 HAL_SMARTCARD_Receive

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Receive (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: Pointer to data buffer • Size: Amount of data to be received • Timeout: Specify timeout value
Return values	<ul style="list-style-type: none"> • HAL status

38.3.11 HAL_SMARTCARD_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit_IT (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Sends an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

38.3.12 HAL_SMARTCARD_Receive_IT

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Receive_IT (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Receives an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: Pointer to data buffer • Size: Amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status

38.3.13 HAL_SMARTCARD_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Transmit_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Sends an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

38.3.14 HAL_SMARTCARD_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_SMARTCARD_Receive_DMA (SMARTCARD_HandleTypeDef * hsc, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module. • pData: Pointer to data buffer • Size: Amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the SMARTCARD parity is enabled (PCE = 1) the data received contain the parity bit.

38.3.15 HAL_SMARTCARD_IRQHandler

Function Name	void HAL_SMARTCARD_IRQHandler (SMARTCARD_HandleTypeDef * hsc)
Function Description	This function handles SMARTCARD interrupt request.
Parameters	<ul style="list-style-type: none"> • hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- None

38.3.16 HAL_SMARTCARD_TxCpltCallback

Function Name **void HAL_SMARTCARD_TxCpltCallback (SMARTCARD_HandleTypeDef * hsc)**

Function Description Tx Transfer completed callback.

Parameters

- **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- None

38.3.17 HAL_SMARTCARD_RxCpltCallback

Function Name **void HAL_SMARTCARD_RxCpltCallback (SMARTCARD_HandleTypeDef * hsc)**

Function Description Rx Transfer completed callback.

Parameters

- **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- None

38.3.18 HAL_SMARTCARD_ErrorCallback

Function Name **void HAL_SMARTCARD_ErrorCallback (SMARTCARD_HandleTypeDef * hsc)**

Function Description SMARTCARD error callback.

Parameters

- **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- None

38.3.19 HAL_SMARTCARD_GetState

Function Name **HAL_SMARTCARD_StateTypeDef HAL_SMARTCARD_GetState (SMARTCARD_HandleTypeDef * hsc)**

Function Description Returns the SMARTCARD state.

Parameters

- **hsc:** Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.

Return values

- HAL state

38.3.20 HAL_SMARTCARD_GetError

Function Name **uint32_t HAL_SMARTCARD_GetError (SMARTCARD_HandleTypeDef * hsc)**

Function Description	Return the SMARTCARD error code.
Parameters	<ul style="list-style-type: none"> hsc: Pointer to a SMARTCARD_HandleTypeDef structure that contains the configuration information for the specified SMARTCARD module.
Return values	<ul style="list-style-type: none"> SMARTCARD Error Code

38.4 SMARTCARD Firmware driver defines

38.4.1 SMARTCARD

SMARTCARD Clock Phase

SMARTCARD_PHASE_1EDGE

SMARTCARD_PHASE_2EDGE

SMARTCARD Clock Polarity

SMARTCARD_POLARITY_LOW

SMARTCARD_POLARITY_HIGH

SMARTCARD DMA requests

SMARTCARD_DMAREQ_TX

SMARTCARD_DMAREQ_RX

SMARTCARD Error Codes

HAL_SMARTCARD_ERROR_NONE No error

HAL_SMARTCARD_ERROR_PE Parity error

HAL_SMARTCARD_ERROR_NE Noise error

HAL_SMARTCARD_ERROR_FE frame error

HAL_SMARTCARD_ERROR_ORE Overrun error

HAL_SMARTCARD_ERROR_DMA DMA transfer error

SMARTCARD Exported Macros

__HAL_SMARTCARD_RESET_HANDLE_STATE

Description:

- Reset SMARTCARD handle state.

Parameters:

- __HANDLE__**: specifies the SMARTCARD Handle.
SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

__HAL_SMARTCARD_FLUSH_DRREGISTER

Description:

- Flush the Smartcard DR register.

`__HAL_SMARTCARD_GET_FLAG`

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.
SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Check whether the specified Smartcard flag is set or not.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.
SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `SMARTCARD_FLAG_TXE`: Transmit data register empty flag
 - `SMARTCARD_FLAG_TC`: Transmission Complete flag
 - `SMARTCARD_FLAG_RXNE`: Receive data register not empty flag
 - `SMARTCARD_FLAG_IDLE`: Idle Line detection flag
 - `SMARTCARD_FLAG_ORE`: OverRun Error flag
 - `SMARTCARD_FLAG_NE`: Noise Error flag
 - `SMARTCARD_FLAG_FE`: Framing Error flag
 - `SMARTCARD_FLAG_PE`: Parity Error flag

Return value:

- The: new state of `__FLAG__` (TRUE or FALSE).

`__HAL_SMARTCARD_CLEAR_FLAG`

Description:

- Clear the specified Smartcard pending flags.

Parameters:

- `__HANDLE__`: specifies the

SMARTCARD Handle.
SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

- **__FLAG__**: specifies the flag to check. This parameter can be any combination of the following values:
 - SMARTCARD_FLAG_TC: Transmission Complete flag.
 - SMARTCARD_FLAG_RXNE: Receive data register not empty flag.

Return value:

- None
- None

Notes:

- PE (Parity error), FE (Framing error), NE (Noise error) and ORE (OverRun error) flags are cleared by software sequence: a read operation to USART_SR register followed by a read operation to USART_DR register. RXNE flag can be also cleared by a read to the USART_DR register. TC flag can be also cleared by software sequence: a read operation to USART_SR register followed by a write operation to USART_DR register. TXE flag is cleared only by a write to the USART_DR register.

__HAL_SMARTCARD_CLEAR_PFLAG

Description:

- Clear the SMARTCARD PE pending flag.

Parameters:

- **__HANDLE__**: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

__HAL_SMARTCARD_CLEAR_FEFLAG

Description:

- Clear the SMARTCARD FE pending flag.

__HAL_SMARTCARD_CLEAR_NEFLAG

Parameters:

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Clear the SMARTCARD NE pending flag.

Parameters:

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Clear the SMARTCARD ORE pending flag.

Parameters:

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Clear the SMARTCARD IDLE pending flag.

Parameters:

- `__HANDLE__`: specifies the USART Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Enable the specified SmartCard

__HAL_SMARTCARD_ENABLE_IT

interrupt.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.
SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__INTERRUPT__`: specifies the SMARTCARD interrupt to enable. This parameter can be one of the following values:
 - `SMARTCARD_IT_TXE`: Transmit Data Register empty interrupt
 - `SMARTCARD_IT_TC`: Transmission complete interrupt
 - `SMARTCARD_IT_RXNE`: Receive Data register not empty interrupt
 - `SMARTCARD_IT_IDLE`: Idle line detection interrupt
 - `SMARTCARD_IT_PE`: Parity Error interrupt
 - `SMARTCARD_IT_ERR`: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

Description:

- Disable the specified SmartCard interrupts.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.
SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__INTERRUPT__`: specifies the SMARTCARD interrupt to disable. This parameter can be one of the following values:
 - `SMARTCARD_IT_TXE`: Transmit Data Register empty interrupt
 - `SMARTCARD_IT_TC`: Transmission complete interrupt

`__HAL_SMARTCARD_DISABLE_IT`

- SMARTCARD_IT_RXNE: Receive Data register not empty interrupt
- SMARTCARD_IT_IDLE: Idle line detection interrupt
- SMARTCARD_IT_PE: Parity Error interrupt
- SMARTCARD_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

__HAL_SMARTCARD_GET_IT_SOURCE

Description:

- Check whether the specified SmartCard interrupt has occurred or not.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- __IT__: specifies the SMARTCARD interrupt source to check. This parameter can be one of the following values:
 - SMARTCARD_IT_TXE: Transmit Data Register empty interrupt
 - SMARTCARD_IT_TC: Transmission complete interrupt
 - SMARTCARD_IT_RXNE: Receive Data register not empty interrupt
 - SMARTCARD_IT_IDLE: Idle line detection interrupt
 - SMARTCARD_IT_ERR: Error interrupt
 - SMARTCARD_IT_PE: Parity Error interrupt

Return value:

- The: new state of __IT__ (TRUE or FALSE).

__HAL_SMARTCARD_ONE_BIT_SAMPLE_ENABLE

Description:

- Enables the SMARTCARD one bit sample method.

Parameters:

- __HANDLE__: specifies the SMARTCARD Handle.

`__HAL_SMARTCARD_ONE_BIT_SAMPLE_DISABLE`

Return value:

- None

Description:

- Disables the SMARTCARD one bit sample method.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle.

Return value:

- None

Description:

- Enable the USART associated to the SMARTCARD Handle.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Disable the USART associated to the SMARTCARD Handle.

Parameters:

- `__HANDLE__`: specifies the SMARTCARD Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Enable the SmartCard DMA request.

Parameters:

- `__HANDLE__`: specifies the SmartCard Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).

`__HAL_SMARTCARD_ENABLE`

`__HAL_SMARTCARD_DISABLE`

`__HAL_SMARTCARD_DMA_REQUEST_ENABLE`

`__HAL_SMARTCARD_DMA_REQUEST_DISABLE`

- `__REQUEST__`: specifies the SmartCard DMA request. This parameter can be one of the following values:
 - `SMARTCARD_DMAREQ_TX`: SmartCard DMA transmit request
 - `SMARTCARD_DMAREQ_RX`: SmartCard DMA receive request

Return value:

- None

Description:

- Disable the SmartCard DMA request.

Parameters:

- `__HANDLE__`: specifies the SmartCard Handle. SMARTCARD Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__REQUEST__`: specifies the SmartCard DMA request. This parameter can be one of the following values:
 - `SMARTCARD_DMAREQ_TX`: SmartCard DMA transmit request
 - `SMARTCARD_DMAREQ_RX`: SmartCard DMA receive request

Return value:

- None

SMARTCARD Flags

`SMARTCARD_FLAG_TXE`
`SMARTCARD_FLAG_TC`
`SMARTCARD_FLAG_RXNE`
`SMARTCARD_FLAG_IDLE`
`SMARTCARD_FLAG_ORE`
`SMARTCARD_FLAG_NE`
`SMARTCARD_FLAG_FE`
`SMARTCARD_FLAG_PE`

SMARTCARD Interrupts Definition

`SMARTCARD_IT_PE`

SMARTCARD_IT_TXE

SMARTCARD_IT_TC

SMARTCARD_IT_RXNE

SMARTCARD_IT_IDLE

SMARTCARD_IT_ERR

SMARTCARD Last Bit

SMARTCARD_LASTBIT_DISABLE

SMARTCARD_LASTBIT_ENABLE

SMARTCARD Mode

SMARTCARD_MODE_RX

SMARTCARD_MODE_TX

SMARTCARD_MODE_TX_RX

SMARTCARD NACK State

SMARTCARD_NACK_ENABLE

SMARTCARD_NACK_DISABLE

SMARTCARD One Bit Sampling Method

SMARTCARD_ONE_BIT_SAMPLE_DISABLE

SMARTCARD_ONE_BIT_SAMPLE_ENABLE

SMARTCARD Parity

SMARTCARD_PARITY_EVEN

SMARTCARD_PARITY_ODD

SMARTCARD Prescaler

SMARTCARD_PRESCALER_SYSCLK_DIV2	SYSCLK divided by 2
SMARTCARD_PRESCALER_SYSCLK_DIV4	SYSCLK divided by 4
SMARTCARD_PRESCALER_SYSCLK_DIV6	SYSCLK divided by 6
SMARTCARD_PRESCALER_SYSCLK_DIV8	SYSCLK divided by 8
SMARTCARD_PRESCALER_SYSCLK_DIV10	SYSCLK divided by 10
SMARTCARD_PRESCALER_SYSCLK_DIV12	SYSCLK divided by 12
SMARTCARD_PRESCALER_SYSCLK_DIV14	SYSCLK divided by 14
SMARTCARD_PRESCALER_SYSCLK_DIV16	SYSCLK divided by 16
SMARTCARD_PRESCALER_SYSCLK_DIV18	SYSCLK divided by 18
SMARTCARD_PRESCALER_SYSCLK_DIV20	SYSCLK divided by 20
SMARTCARD_PRESCALER_SYSCLK_DIV22	SYSCLK divided by 22
SMARTCARD_PRESCALER_SYSCLK_DIV24	SYSCLK divided by 24
SMARTCARD_PRESCALER_SYSCLK_DIV26	SYSCLK divided by 26
SMARTCARD_PRESCALER_SYSCLK_DIV28	SYSCLK divided by 28

SMARTCARD_PRESCALER_SYSCLOCK_DIV30	SYSCLOCK divided by 30
SMARTCARD_PRESCALER_SYSCLOCK_DIV32	SYSCLOCK divided by 32
SMARTCARD_PRESCALER_SYSCLOCK_DIV34	SYSCLOCK divided by 34
SMARTCARD_PRESCALER_SYSCLOCK_DIV36	SYSCLOCK divided by 36
SMARTCARD_PRESCALER_SYSCLOCK_DIV38	SYSCLOCK divided by 38
SMARTCARD_PRESCALER_SYSCLOCK_DIV40	SYSCLOCK divided by 40
SMARTCARD_PRESCALER_SYSCLOCK_DIV42	SYSCLOCK divided by 42
SMARTCARD_PRESCALER_SYSCLOCK_DIV44	SYSCLOCK divided by 44
SMARTCARD_PRESCALER_SYSCLOCK_DIV46	SYSCLOCK divided by 46
SMARTCARD_PRESCALER_SYSCLOCK_DIV48	SYSCLOCK divided by 48
SMARTCARD_PRESCALER_SYSCLOCK_DIV50	SYSCLOCK divided by 50
SMARTCARD_PRESCALER_SYSCLOCK_DIV52	SYSCLOCK divided by 52
SMARTCARD_PRESCALER_SYSCLOCK_DIV54	SYSCLOCK divided by 54
SMARTCARD_PRESCALER_SYSCLOCK_DIV56	SYSCLOCK divided by 56
SMARTCARD_PRESCALER_SYSCLOCK_DIV58	SYSCLOCK divided by 58
SMARTCARD_PRESCALER_SYSCLOCK_DIV60	SYSCLOCK divided by 60
SMARTCARD_PRESCALER_SYSCLOCK_DIV62	SYSCLOCK divided by 62

SMARTCARD Private Macros

SMARTCARD_CR1_REG_INDEX

SMARTCARD_CR3_REG_INDEX

SMARTCARD_DIV

SMARTCARD_DIVMANT

SMARTCARD_DIVFRAQ

SMARTCARD_BRR

IS_SMARTCARD_BAUDRATE

The maximum Baud Rate is derived from the maximum clock on APB (i.e. 32 MHz) divided by the smallest oversampling used on the USART (i.e. 16) `__BAUDRATE__`; Baud rate set by the configuration function. Return : TRUE or FALSE

IS_SMARTCARD_WORD_LENGTH

IS_SMARTCARD_STOPBITS

IS_SMARTCARD_PARITY

IS_SMARTCARD_MODE

IS_SMARTCARD_POLARITY

IS_SMARTCARD_PHASE

IS_SMARTCARD_LASTBIT

IS_SMARTCARD_ONE_BIT_SAMPLE

IS_SMARTCARD_NACK_STATE

IS_SMARTCARD_PRESCALER

SMARTCARD_IT_MASK

SMARTCARD Number of Stop Bits

SMARTCARD_STOPBITS_0_5

SMARTCARD_STOPBITS_1_5

SMARTCARD Word Length

SMARTCARD_WORDLENGTH_9B

39 HAL SPI Generic Driver

39.1 HAL SPI Generic Driver

39.2 SPI Firmware driver registers structures

39.2.1 SPI_InitTypeDef

Data Fields

- *uint32_t Mode*
- *uint32_t Direction*
- *uint32_t DataSize*
- *uint32_t CLKPolarity*
- *uint32_t CLKPhase*
- *uint32_t NSS*
- *uint32_t BaudRatePrescaler*
- *uint32_t FirstBit*
- *uint32_t TIMode*
- *uint32_t CRCCalculation*
- *uint32_t CRCPolynomial*

Field Documentation

- *uint32_t SPI_InitTypeDef::Mode*
Specifies the SPI operating mode. This parameter can be a value of [SPI_mode](#)
- *uint32_t SPI_InitTypeDef::Direction*
Specifies the SPI Directional mode state. This parameter can be a value of [SPI_Direction_mode](#)
- *uint32_t SPI_InitTypeDef::DataSize*
Specifies the SPI data size. This parameter can be a value of [SPI_data_size](#)
- *uint32_t SPI_InitTypeDef::CLKPolarity*
Specifies the serial clock steady state. This parameter can be a value of [SPI_Clock_Polarity](#)
- *uint32_t SPI_InitTypeDef::CLKPhase*
Specifies the clock active edge for the bit capture. This parameter can be a value of [SPI_Clock_Phase](#)
- *uint32_t SPI_InitTypeDef::NSS*
Specifies whether the NSS signal is managed by hardware (NSS pin) or by software using the SSI bit. This parameter can be a value of [SPI_Slave_Select_management](#)
- *uint32_t SPI_InitTypeDef::BaudRatePrescaler*
Specifies the Baud Rate prescaler value which will be used to configure the transmit and receive SCK clock. This parameter can be a value of [SPI_BaudRate_Prescaler](#)
Note: The communication clock is derived from the master clock. The slave clock does not need to be set
- *uint32_t SPI_InitTypeDef::FirstBit*
Specifies whether data transfers start from MSB or LSB bit. This parameter can be a value of [SPI_MSB_LSB_transmission](#)

- **`uint32_t SPI_InitTypeDef::TIMode`**
Specifies if the TI mode is enabled or not. This parameter can be a value of [SPI_TI_mode](#)
- **`uint32_t SPI_InitTypeDef::CRCCalculation`**
Specifies if the CRC calculation is enabled or not. This parameter can be a value of [SPI_CRC_Calculation](#)
- **`uint32_t SPI_InitTypeDef::CRCPolynomial`**
Specifies the polynomial used for the CRC calculation. This parameter must be a number between Min_Data = 0 and Max_Data = 65535

39.2.2 `__SPI_HandleTypeDef`

Data Fields

- **`SPI_TypeDef * Instance`**
- **`SPI_InitTypeDef Init`**
- **`uint8_t * pTxBuffPtr`**
- **`uint16_t TxXferSize`**
- **`uint16_t TxXferCount`**
- **`uint8_t * pRxBuffPtr`**
- **`uint16_t RxXferSize`**
- **`uint16_t RxXferCount`**
- **`DMA_HandleTypeDef * hdmatx`**
- **`DMA_HandleTypeDef * hdmarx`**
- **`void(* RxISR`**
- **`void(* TxISR`**
- **`HAL_LockTypeDef Lock`**
- **`__IO HAL_SPI_StateTypeDef State`**
- **`__IO uint32_t ErrorCode`**

Field Documentation

- **`SPI_TypeDef* __SPI_HandleTypeDef::Instance`**
- **`SPI_InitTypeDef __SPI_HandleTypeDef::Init`**
- **`uint8_t* __SPI_HandleTypeDef::pTxBuffPtr`**
- **`uint16_t __SPI_HandleTypeDef::TxXferSize`**
- **`uint16_t __SPI_HandleTypeDef::TxXferCount`**
- **`uint8_t* __SPI_HandleTypeDef::pRxBuffPtr`**
- **`uint16_t __SPI_HandleTypeDef::RxXferSize`**
- **`uint16_t __SPI_HandleTypeDef::RxXferCount`**
- **`DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmatx`**
- **`DMA_HandleTypeDef* __SPI_HandleTypeDef::hdmarx`**
- **`void(* __SPI_HandleTypeDef::RxISR)(struct __SPI_HandleTypeDef *hspi)`**
- **`void(* __SPI_HandleTypeDef::TxISR)(struct __SPI_HandleTypeDef *hspi)`**
- **`HAL_LockTypeDef __SPI_HandleTypeDef::Lock`**
- **`__IO HAL_SPI_StateTypeDef __SPI_HandleTypeDef::State`**
- **`__IO uint32_t __SPI_HandleTypeDef::ErrorCode`**

39.3 SPI Firmware driver API description

39.3.1 How to use this driver

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example: SPI_HandleTypeDef hspi;
2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit ()API:
 - a. Enable the SPIx interface clock
 - b. SPI pins configuration
 - Enable the clock for the SPI GPIOs
 - Configure these SPI pins as alternate function push-pull
 - c. NVIC configuration if you need to use interrupt process
 - Configure the SPIx interrupt priority
 - Enable the NVIC SPI IRQ handle
 - d. DMA Configuration if you need to use DMA process
 - Declare a DMA_HandleTypeDef handle structure for the transmit or receive Channel
 - Enable the DMAx clock
 - Configure the DMA handle parameters
 - Configure the DMA Tx or Rx Channel
 - Associate the initialized hdma_tx(or _rx) handle to the hspi DMA Tx (or Rx) handle
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx Channel
3. Program the Mode, Direction , Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL_SPI_Init() API:
 - This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit() API.

Circular mode restriction:

1. The DMA circular mode cannot be used when the SPI is configured in these modes:
 - a. Master 2Lines RxOnly
 - b. Master 1Line Rx
2. The CRC feature is not managed when the DMA circular mode is enabled
3. When the SPI DMA Pause/Stop features are used, we must use the following APIs the HAL_SPI_DMAPause()/ HAL_SPI_DMAStop() only under the SPI callbacks

39.3.2 Initialization and de-initialization functions

This subsection provides a set of functions allowing to initialize and de-initialize the SPIx peripheral:

- User must implement HAL_SPI_MspInit() function in which he configures all related peripherals resources (CLOCK, GPIO, DMA, IT and NVIC).
- Call the function HAL_SPI_Init() to configure the selected device with the selected configuration:
 - Mode
 - Direction
 - Data Size
 - Clock Polarity and Phase
 - NSS Management
 - BaudRate Prescaler
 - FirstBit

- TMode
- CRC Calculation
- CRC Polynomial if CRC enabled
- Call the function HAL_SPI_DeInit() to restore the default configuration of the selected SPIx peripheral.

This section contains the following APIs:

- [HAL_SPI_Init\(\)](#)
- [HAL_SPI_DeInit\(\)](#)
- [HAL_SPI_MspltInit\(\)](#)
- [HAL_SPI_MspltDeInit\(\)](#)

39.3.3 IO operation functions

The SPI supports master and slave mode :

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA, These APIs return the HAL status. The end of the data processing will be indicated through the dedicated SPI IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_SPI_TxCpltCallback(), HAL_SPI_RxCpltCallback() and HAL_SPI_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive process The HAL_SPI_ErrorCallback() user callback will be executed when a communication error is detected
2. APIs provided for these 2 transfer modes (Blocking mode or Non blocking mode using either Interrupt or DMA) exist for 1Line (simplex) and 2Lines (full duplex) modes.

This section contains the following APIs:

- [HAL_SPI_Transmit\(\)](#)
- [HAL_SPI_Receive\(\)](#)
- [HAL_SPI_TransmitReceive\(\)](#)
- [HAL_SPI_Transmit_IT\(\)](#)
- [HAL_SPI_Receive_IT\(\)](#)
- [HAL_SPI_TransmitReceive_IT\(\)](#)
- [HAL_SPI_Transmit_DMA\(\)](#)
- [HAL_SPI_Receive_DMA\(\)](#)
- [HAL_SPI_TransmitReceive_DMA\(\)](#)
- [HAL_SPI_DMABase\(\)](#)
- [HAL_SPI_DMAResume\(\)](#)
- [HAL_SPI_DMABaseStop\(\)](#)
- [HAL_SPI_IRQHandler\(\)](#)
- [HAL_SPI_TxCpltCallback\(\)](#)
- [HAL_SPI_RxCpltCallback\(\)](#)
- [HAL_SPI_TxRxCpltCallback\(\)](#)
- [HAL_SPI_TxHalfCpltCallback\(\)](#)
- [HAL_SPI_RxHalfCpltCallback\(\)](#)
- [HAL_SPI_TxRxHalfCpltCallback\(\)](#)
- [HAL_SPI_ErrorCallback\(\)](#)

39.3.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to control the SPI.

- `HAL_SPI_GetState()` API can be helpful to check in run-time the state of the SPI peripheral
- `HAL_SPI_GetError()` check in run-time Errors occurring during communication

This section contains the following APIs:

- [*HAL_SPI_GetState\(\)*](#)
- [*HAL_SPI_GetError\(\)*](#)

39.3.5 HAL_SPI_Init

Function Name	HAL_StatusTypeDef HAL_SPI_Init (SPI_HandleTypeDef * hspi)
Function Description	Initializes the SPI according to the specified parameters in the <code>SPI_InitTypeDef</code> and create the associated handle.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • HAL status

39.3.6 HAL_SPI_DeInit

Function Name	HAL_StatusTypeDef HAL_SPI_DeInit (SPI_HandleTypeDef * hspi)
Function Description	DeInitializes the SPI peripheral.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • HAL status

39.3.7 HAL_SPI_MspInit

Function Name	void HAL_SPI_MspInit (SPI_HandleTypeDef * hspi)
Function Description	SPI MSP Init.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

39.3.8 HAL_SPI_MspDeInit

Function Name	void HAL_SPI_MspDeInit (SPI_HandleTypeDef * hspi)
Function Description	SPI MSP DeInit.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a <code>SPI_HandleTypeDef</code> structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> • None

39.3.9 HAL_SPI_Transmit

Function Name	HAL_StatusTypeDef HAL_SPI_Transmit (SPI_HandleTypeDef *
---------------	--

hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)

Function Description Transmit an amount of data in blocking mode.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent
- **Timeout:** Timeout duration

Return values

- HAL status

39.3.10 HAL_SPI_Receive

Function Name **HAL_StatusTypeDef HAL_SPI_Receive (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size, uint32_t Timeout)**

Function Description Receive an amount of data in blocking mode.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent
- **Timeout:** Timeout duration

Return values

- HAL status

39.3.11 HAL_SPI_TransmitReceive

Function Name **HAL_StatusTypeDef HAL_SPI_TransmitReceive (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)**

Function Description Transmit and Receive an amount of data in blocking mode.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pTxData:** pointer to transmission data buffer
- **pRxData:** pointer to reception data buffer to be
- **Size:** amount of data to be sent
- **Timeout:** Timeout duration

Return values

- HAL status

39.3.12 HAL_SPI_Transmit_IT

Function Name **HAL_StatusTypeDef HAL_SPI_Transmit_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)**

Function Description Transmit an amount of data in no-blocking mode with Interrupt.

Parameters

- **hspi:** pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
- **pData:** pointer to data buffer
- **Size:** amount of data to be sent

Return values

- HAL status

39.3.13 HAL_SPI_Receive_IT

Function Name	HAL_StatusTypeDef HAL_SPI_Receive_IT (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

39.3.14 HAL_SPI_TransmitReceive_IT

Function Name	HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Transmit and Receive an amount of data in no-blocking mode with Interrupt.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pTxData: pointer to transmission data buffer • pRxData: pointer to reception data buffer to be • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

39.3.15 HAL_SPI_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_SPI_Transmit_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
Function Description	Transmit an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

39.3.16 HAL_SPI_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_SPI_Receive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pData, uint16_t Size)
Function Description	Receive an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pData: pointer to data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the CRC feature is enabled the pData Length must be Size + 1.

39.3.17 HAL_SPI_TransmitReceive_DMA

Function Name	HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA (SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Transmit and Receive an amount of data in no-blocking mode with DMA.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module. • pTxData: pointer to transmission data buffer • pRxData: pointer to reception data buffer • Size: amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the CRC feature is enabled the pRxData Length must be Size + 1

39.3.18 HAL_SPI_DMAPause

Function Name	HAL_StatusTypeDef HAL_SPI_DMAPause (SPI_HandleTypeDef * hspi)
Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none"> • HAL status

39.3.19 HAL_SPI_DMAResume

Function Name	HAL_StatusTypeDef HAL_SPI_DMAResume (SPI_HandleTypeDef * hspi)
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified SPI module.
Return values	<ul style="list-style-type: none"> • HAL status

39.3.20 HAL_SPI_DMAStop

Function Name	HAL_StatusTypeDef HAL_SPI_DMAStop (SPI_HandleTypeDef * hspi)
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> • hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> • HAL status

39.3.21 HAL_SPI_IRQHandler

Function Name	void HAL_SPI_IRQHandler (SPI_HandleTypeDef * hspi)
Function Description	This function handles SPI interrupt request.

Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• HAL status

39.3.22 HAL_SPI_TxCpltCallback

Function Name	void HAL_SPI_TxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None

39.3.23 HAL_SPI_RxCpltCallback

Function Name	void HAL_SPI_RxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None

39.3.24 HAL_SPI_TxRxCpltCallback

Function Name	void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None

39.3.25 HAL_SPI_TxHalfCpltCallback

Function Name	void HAL_SPI_TxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None

39.3.26 HAL_SPI_RxHalfCpltCallback

Function Name	void HAL_SPI_RxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none">• None

39.3.27 HAL_SPI_TxRxHalfCpltCallback

Function Name	void HAL_SPI_TxRxHalfCpltCallback (SPI_HandleTypeDef * hspi)
Function Description	Tx and Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> None

39.3.28 HAL_SPI_ErrorCallback

Function Name	void HAL_SPI_ErrorCallback (SPI_HandleTypeDef * hspi)
Function Description	SPI error callbacks.
Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> None

39.3.29 HAL_SPI_GetState

Function Name	HAL_SPI_StateTypeDef HAL_SPI_GetState (SPI_HandleTypeDef * hspi)
Function Description	Return the SPI state.
Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> HAL state

39.3.30 HAL_SPI_GetError

Function Name	uint32_t HAL_SPI_GetError (SPI_HandleTypeDef * hspi)
Function Description	Return the SPI error code.
Parameters	<ul style="list-style-type: none"> hspi: pointer to a SPI_HandleTypeDef structure that contains the configuration information for SPI module.
Return values	<ul style="list-style-type: none"> SPI Error Code

39.4 SPI Firmware driver defines**39.4.1 SPI*****SPI BaudRate Prescaler***

SPI_BAUDRATEPRESCALER_2
 SPI_BAUDRATEPRESCALER_4
 SPI_BAUDRATEPRESCALER_8
 SPI_BAUDRATEPRESCALER_16
 SPI_BAUDRATEPRESCALER_32
 SPI_BAUDRATEPRESCALER_64

SPI_BAUDRATEPRESCALER_128

SPI_BAUDRATEPRESCALER_256

IS_SPI_BAUDRATE_PRESCALER

SPI Clock Phase

SPI_PHASE_1EDGE

SPI_PHASE_2EDGE

IS_SPI_CPHA

SPI Clock Polarity

SPI_POLARITY_LOW

SPI_POLARITY_HIGH

IS_SPI_CPOL

SPI CRC Calculation

SPI_CRCCALCULATION_DISABLE

SPI_CRCCALCULATION_ENABLE

IS_SPI_CRC_CALCULATION

IS_SPI_CRC_POLYNOMIAL

SPI data size

SPI_DATASIZE_8BIT

SPI_DATASIZE_16BIT

IS_SPI_DATASIZE

SPI Direction mode

SPI_DIRECTION_2LINES

SPI_DIRECTION_2LINES_RXONLY

SPI_DIRECTION_1LINE

IS_SPI_DIRECTION_MODE

IS_SPI_DIRECTION_2LINES_OR_1LINE

IS_SPI_DIRECTION_2LINES

SPI Error Codes

HAL_SPI_ERROR_NONE No error

HAL_SPI_ERROR_MODF MODF error

HAL_SPI_ERROR_CRC CRC error

HAL_SPI_ERROR_OVR OVR error

HAL_SPI_ERROR_FRE FRE error

HAL_SPI_ERROR_DMA DMA transfer error

HAL_SPI_ERROR_FLAG Flag: RXNE, TXE, BSY

SPI Exported Macros

__HAL_SPI_RESET_HANDLE_STATE**Description:**

- Reset SPI handle state.

Parameters:

- **__HANDLE__**: specifies the SPI handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

__HAL_SPI_ENABLE_IT**Description:**

- Enable or disable the specified SPI interrupts.

Parameters:

- **__HANDLE__**: specifies the SPI handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- **__INTERRUPT__**: specifies the interrupt source to enable or disable. This parameter can be one of the following values:
 - SPI_IT_TXE: Tx buffer empty interrupt enable
 - SPI_IT_RXNE: RX buffer not empty interrupt enable
 - SPI_IT_ERR: Error interrupt enable

Return value:

- None

__HAL_SPI_DISABLE_IT**__HAL_SPI_GET_IT_SOURCE****Description:**

- Check if the specified SPI interrupt source is enabled or disabled.

Parameters:

- **__HANDLE__**: specifies the SPI handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- **__INTERRUPT__**: specifies the SPI interrupt source to check. This parameter can be one of the following values:
 - SPI_IT_TXE: Tx buffer empty interrupt enable
 - SPI_IT_RXNE: RX buffer not empty interrupt enable
 - SPI_IT_ERR: Error interrupt enable

Return value:

- The: new state of **__IT__** (TRUE or FALSE).

__HAL_SPI_GET_FLAG**Description:**

- Check whether the specified SPI flag is set or not.

Parameters:

- `__HANDLE__`: specifies the SPI handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `SPI_FLAG_RXNE`: Receive buffer not empty flag
 - `SPI_FLAG_TXE`: Transmit buffer empty flag
 - `SPI_FLAG_CRCERR`: CRC error flag
 - `SPI_FLAG_MODF`: Mode fault flag
 - `SPI_FLAG_OVR`: Overrun flag
 - `SPI_FLAG_BSY`: Busy flag
 - `SPI_FLAG_FRE`: Frame format error flag

Return value:

- The: new state of `__FLAG__` (TRUE or FALSE).

`__HAL_SPI_CLEAR_CRCERRFLAG`**Description:**

- Clear the SPI CRCERR pending flag.

Parameters:

- `__HANDLE__`: specifies the SPI handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

`__HAL_SPI_CLEAR_MODFFLAG`**Description:**

- Clear the SPI MODF pending flag.

Parameters:

- `__HANDLE__`: specifies the SPI handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

`__HAL_SPI_CLEAR_OVRFLAG`**Description:**

- Clear the SPI OVR pending flag.

Parameters:

- `__HANDLE__`: specifies the SPI handle.
This parameter can be SPI where x: 1, 2, or

3 to select the SPI peripheral.

Return value:

- None

Description:

- Clear the SPI FRE pending flag.

Parameters:

- `__HANDLE__`: specifies the SPI handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

Description:

- Enables the SPI.

Parameters:

- `__HANDLE__`: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

Description:

- Disables the SPI.

Parameters:

- `__HANDLE__`: specifies the SPI Handle.
This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

`__HAL_SPI_CLEAR_FREFLAG`

`__HAL_SPI_ENABLE`

`__HAL_SPI_DISABLE`

SPI Flag definition

`SPI_FLAG_RXNE`

`SPI_FLAG_TXE`

`SPI_FLAG_CRCERR`

`SPI_FLAG_MODF`

`SPI_FLAG_OVR`

`SPI_FLAG_BSY`

`SPI_FLAG_FRE`

SPI Interrupt configuration definition

`SPI_IT_TXE`

`SPI_IT_RXNE`

SPI_IT_ERR

SPI mode

SPI_MODE_SLAVE

SPI_MODE_MASTER

IS_SPI_MODE

SPI MSB LSB transmission

SPI_FIRSTBIT_MSB

SPI_FIRSTBIT_LSB

IS_SPI_FIRST_BIT

SPI Private Constants

SPI_TIMEOUT_VALUE

SPI Private Macros

SPI_1LINE_TX

Description:

- Sets the SPI transmit-only mode.

Parameters:

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

SPI_1LINE_RX

Description:

- Sets the SPI receive-only mode.

Parameters:

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

SPI_RESET_CRC

Description:

- Resets the CRC calculation of the SPI.

Parameters:

- `__HANDLE__`: specifies the SPI Handle. This parameter can be SPI where x: 1, 2, or 3 to select the SPI peripheral.

Return value:

- None

SPI Slave Select management

SPI_NSS_SOFT

SPI_NSS_HARD_INPUT

SPI_NSS_HARD_OUTPUT

IS_SPI_NSS

SPI TI mode

SPI_TIMODE_DISABLE

SPI_TIMODE_ENABLE

IS_SPI_TIMODE

40 HAL SRAM Generic Driver

40.1 HAL SRAM Generic Driver

40.2 SRAM Firmware driver registers structures

40.2.1 SRAM_HandleTypeDef

Data Fields

- *FSMC_NORSRAM_TypeDef * Instance*
- *FSMC_NORSRAM_EXTENDED_TypeDef * Extended*
- *FSMC_NORSRAM_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_SRAM_StateTypeDef State*
- *DMA_HandleTypeDef * hdma*

Field Documentation

- *FSMC_NORSRAM_TypeDef* SRAM_HandleTypeDef::Instance*
Register base address
- *FSMC_NORSRAM_EXTENDED_TypeDef* SRAM_HandleTypeDef::Extended*
Extended mode register base address
- *FSMC_NORSRAM_InitTypeDef SRAM_HandleTypeDef::Init*
SRAM device control configuration parameters
- *HAL_LockTypeDef SRAM_HandleTypeDef::Lock*
SRAM locking object
- *__IO HAL_SRAM_StateTypeDef SRAM_HandleTypeDef::State*
SRAM device access state
- *DMA_HandleTypeDef* SRAM_HandleTypeDef::hdma*
Pointer DMA handler

40.3 SRAM Firmware driver API description

40.3.1 How to use this driver

This driver is a generic layered driver which contains a set of APIs used to control SRAM memories. It uses the FSMC layer functions to interface with SRAM devices. The following sequence should be followed to configure the FSMC to interface with SRAM/PSRAM memories:

1. Declare a SRAM_HandleTypeDef handle structure, for example:
SRAM_HandleTypeDef hsram; and:
 - Fill the SRAM_HandleTypeDef handle "Init" field with the allowed values of the structure member.
 - Fill the SRAM_HandleTypeDef handle "Instance" field with a predefined base register instance for NOR or SRAM device
 - Fill the SRAM_HandleTypeDef handle "Extended" field with a predefined base register instance for NOR or SRAM extended mode

2. Declare two `FSMC_NORSRAM_TimingTypeDef` structures, for both normal and extended mode timings; for example: `FSMC_NORSRAM_TimingTypeDef Timing` and `FSMC_NORSRAM_TimingTypeDef ExTiming`; and fill its fields with the allowed values of the structure member.
3. Initialize the SRAM Controller by calling the function `HAL_SRAM_Init()`. This function performs the following sequence:
 - a. MSP hardware layer configuration using the function `HAL_SRAM_MspInit()`
 - b. Control register configuration using the FSMC NORSRAM interface function `FSMC_NORSRAM_Init()`
 - c. Timing register configuration using the FSMC NORSRAM interface function `FSMC_NORSRAM_Timing_Init()`
 - d. Extended mode Timing register configuration using the FSMC NORSRAM interface function `FSMC_NORSRAM_Extended_Timing_Init()`
 - e. Enable the SRAM device using the macro `__FSMC_NORSRAM_ENABLE()`
4. At this stage you can perform read/write accesses from/to the memory connected to the NOR/SRAM Bank. You can perform either polling or DMA transfer using the following APIs:
 - `HAL_SRAM_Read()/HAL_SRAM_Write()` for polling read/write access
 - `HAL_SRAM_Read_DMA()/HAL_SRAM_Write_DMA()` for DMA read/write transfer
5. You can also control the SRAM device by calling the control APIs `HAL_SRAM_WriteOperation_Enable()/HAL_SRAM_WriteOperation_Disable()` to respectively enable/disable the SRAM write operation
6. You can continuously monitor the SRAM device HAL state by calling the function `HAL_SRAM_GetState()`

40.3.2 SRAM Initialization and de_initialization functions

This section provides functions allowing to initialize/de-initialize the SRAM memory

This section contains the following APIs:

- [`HAL_SRAM_Init\(\)`](#)
- [`HAL_SRAM_DeInit\(\)`](#)
- [`HAL_SRAM_MspInit\(\)`](#)
- [`HAL_SRAM_MspDeInit\(\)`](#)
- [`HAL_SRAM_DMA_XferCpltCallback\(\)`](#)
- [`HAL_SRAM_DMA_XferErrorCallback\(\)`](#)

40.3.3 SRAM Input and Output functions

This section provides functions allowing to use and control the SRAM memory

This section contains the following APIs:

- [`HAL_SRAM_Read_8b\(\)`](#)
- [`HAL_SRAM_Write_8b\(\)`](#)
- [`HAL_SRAM_Read_16b\(\)`](#)
- [`HAL_SRAM_Write_16b\(\)`](#)
- [`HAL_SRAM_Read_32b\(\)`](#)
- [`HAL_SRAM_Write_32b\(\)`](#)
- [`HAL_SRAM_Read_DMA\(\)`](#)
- [`HAL_SRAM_Write_DMA\(\)`](#)

40.3.4 SRAM Control functions

This subsection provides a set of functions allowing to control dynamically the SRAM interface.

This section contains the following APIs:

- [HAL_SRAM_WriteOperation_Enable\(\)](#)
- [HAL_SRAM_WriteOperation_Disable\(\)](#)

40.3.5 SRAM State functions

This subsection permits to get in run-time the status of the SRAM controller and the data flow.

This section contains the following APIs:

- [HAL_SRAM_GetState\(\)](#)

40.3.6 HAL_SRAM_Init

Function Name	HAL_StatusTypeDef HAL_SRAM_Init (SRAM_HandleTypeDef * hsram, FSMC_NORSRAM_TimingTypeDef * Timing, FSMC_NORSRAM_TimingTypeDef * ExtTiming)
Function Description	Performs the SRAM device initialization sequence.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • Timing: Pointer to SRAM control timing structure • ExtTiming: Pointer to SRAM extended mode timing structure
Return values	<ul style="list-style-type: none"> • HAL status

40.3.7 HAL_SRAM_DeInit

Function Name	HAL_StatusTypeDef HAL_SRAM_DeInit (SRAM_HandleTypeDef * hsram)
Function Description	Performs the SRAM device De-initialization sequence.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • HAL status

40.3.8 HAL_SRAM_MspInit

Function Name	void HAL_SRAM_MspInit (SRAM_HandleTypeDef * hsram)
Function Description	SRAM MSP Init.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • None

40.3.9 HAL_SRAM_MspDeInit

Function Name	void HAL_SRAM_MspDeInit (SRAM_HandleTypeDef * hsram)
Function Description	SRAM MSP DeInit.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • None

40.3.10 HAL_SRAM_DMA_XferCpltCallback

Function Name	void HAL_SRAM_DMA_XferCpltCallback (DMA_HandleTypeDef * hdma)
Function Description	DMA transfer complete callback.
Parameters	<ul style="list-style-type: none"> hdma: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> None

40.3.11 HAL_SRAM_DMA_XferErrorCallback

Function Name	void HAL_SRAM_DMA_XferErrorCallback (DMA_HandleTypeDef * hdma)
Function Description	DMA transfer complete error callback.
Parameters	<ul style="list-style-type: none"> hdma: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> None

40.3.12 HAL_SRAM_Read_8b

Function Name	HAL_StatusTypeDef HAL_SRAM_Read_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pDstBuffer, uint32_t BufferSize)
Function Description	Reads 8-bit buffer from SRAM memory.
Parameters	<ul style="list-style-type: none"> hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. pAddress: Pointer to read start address pDstBuffer: Pointer to destination buffer BufferSize: Size of the buffer to read from memory
Return values	<ul style="list-style-type: none"> HAL status

40.3.13 HAL_SRAM_Write_8b

Function Name	HAL_StatusTypeDef HAL_SRAM_Write_8b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint8_t * pSrcBuffer, uint32_t BufferSize)
Function Description	Writes 8-bit buffer to SRAM memory.
Parameters	<ul style="list-style-type: none"> hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. pAddress: Pointer to write start address pSrcBuffer: Pointer to source buffer to write BufferSize: Size of the buffer to write to memory
Return values	<ul style="list-style-type: none"> HAL status

40.3.14 HAL_SRAM_Read_16b

Function Name	HAL_StatusTypeDef HAL_SRAM_Read_16b (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint16_t
---------------	---

* **pDstBuffer**, **uint32_t BufferSize**)

Function Description Reads 16-bit buffer from SRAM memory.

Parameters

- **hsram**: pointer to a **SRAM_HandleTypeDef** structure that contains the configuration information for SRAM module.
- **pAddress**: Pointer to read start address
- **pDstBuffer**: Pointer to destination buffer
- **BufferSize**: Size of the buffer to read from memory

Return values

- HAL status

40.3.15 HAL_SRAM_Write_16b

Function Name **HAL_StatusTypeDef HAL_SRAM_Write_16b**
(**SRAM_HandleTypeDef * hsram**, **uint32_t * pAddress**, **uint16_t * pSrcBuffer**, **uint32_t BufferSize**)

Function Description Writes 16-bit buffer to SRAM memory.

Parameters

- **hsram**: pointer to a **SRAM_HandleTypeDef** structure that contains the configuration information for SRAM module.
- **pAddress**: Pointer to write start address
- **pSrcBuffer**: Pointer to source buffer to write
- **BufferSize**: Size of the buffer to write to memory

Return values

- HAL status

40.3.16 HAL_SRAM_Read_32b

Function Name **HAL_StatusTypeDef HAL_SRAM_Read_32b**
(**SRAM_HandleTypeDef * hsram**, **uint32_t * pAddress**, **uint32_t * pDstBuffer**, **uint32_t BufferSize**)

Function Description Reads 32-bit buffer from SRAM memory.

Parameters

- **hsram**: pointer to a **SRAM_HandleTypeDef** structure that contains the configuration information for SRAM module.
- **pAddress**: Pointer to read start address
- **pDstBuffer**: Pointer to destination buffer
- **BufferSize**: Size of the buffer to read from memory

Return values

- HAL status

40.3.17 HAL_SRAM_Write_32b

Function Name **HAL_StatusTypeDef HAL_SRAM_Write_32b**
(**SRAM_HandleTypeDef * hsram**, **uint32_t * pAddress**, **uint32_t * pSrcBuffer**, **uint32_t BufferSize**)

Function Description Writes 32-bit buffer to SRAM memory.

Parameters

- **hsram**: pointer to a **SRAM_HandleTypeDef** structure that contains the configuration information for SRAM module.
- **pAddress**: Pointer to write start address
- **pSrcBuffer**: Pointer to source buffer to write
- **BufferSize**: Size of the buffer to write to memory

Return values

- HAL status

40.3.18 HAL_SRAM_Read_DMA

Function Name	HAL_StatusTypeDef HAL_SRAM_Read_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pDstBuffer, uint32_t BufferSize)
Function Description	Reads a Words data from the SRAM memory using DMA transfer.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • pAddress: Pointer to read start address • pDstBuffer: Pointer to destination buffer • BufferSize: Size of the buffer to read from memory
Return values	<ul style="list-style-type: none"> • HAL status

40.3.19 HAL_SRAM_Write_DMA

Function Name	HAL_StatusTypeDef HAL_SRAM_Write_DMA (SRAM_HandleTypeDef * hsram, uint32_t * pAddress, uint32_t * pSrcBuffer, uint32_t BufferSize)
Function Description	Writes a Words data buffer to SRAM memory using DMA transfer.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module. • pAddress: Pointer to write start address • pSrcBuffer: Pointer to source buffer to write • BufferSize: Size of the buffer to write to memory
Return values	<ul style="list-style-type: none"> • HAL status

40.3.20 HAL_SRAM_WriteOperation_Enable

Function Name	HAL_StatusTypeDef HAL_SRAM_WriteOperation_Enable (SRAM_HandleTypeDef * hsram)
Function Description	Enables dynamically SRAM write operation.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • HAL status

40.3.21 HAL_SRAM_WriteOperation_Disable

Function Name	HAL_StatusTypeDef HAL_SRAM_WriteOperation_Disable (SRAM_HandleTypeDef * hsram)
Function Description	Disables dynamically SRAM write operation.
Parameters	<ul style="list-style-type: none"> • hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none"> • HAL status

40.3.22 HAL_SRAM_GetState

Function Name	HAL_SRAM_StateTypeDef HAL_SRAM_GetState (SRAM_HandleTypeDef * hsram)
---------------	--

Function Description	Returns the SRAM controller state.
Parameters	<ul style="list-style-type: none">• hsram: pointer to a SRAM_HandleTypeDef structure that contains the configuration information for SRAM module.
Return values	<ul style="list-style-type: none">• HAL state

40.4 SRAM Firmware driver defines

40.4.1 SRAM

SRAM Exported Macros

<code>__HAL_SRAM_RESET_HANDLE_STATE</code>	Description: <ul style="list-style-type: none">• Reset SRAM handle state. Parameters: <ul style="list-style-type: none">• <code>__HANDLE__</code>: SRAM handle Return value: <ul style="list-style-type: none">• None
--	--

41 HAL TIM Generic Driver

41.1 HAL TIM Generic Driver

41.2 TIM Firmware driver registers structures

41.2.1 TIM_Base_InitTypeDef

Data Fields

- *uint32_t Prescaler*
- *uint32_t CounterMode*
- *uint32_t Period*
- *uint32_t ClockDivision*

Field Documentation

- *uint32_t TIM_Base_InitTypeDef::Prescaler*
Specifies the prescaler value used to divide the TIM clock. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- *uint32_t TIM_Base_InitTypeDef::CounterMode*
Specifies the counter mode. This parameter can be a value of [TIM_Counter_Mode](#)
- *uint32_t TIM_Base_InitTypeDef::Period*
Specifies the period value to be loaded into the active Auto-Reload Register at the next update event. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.
- *uint32_t TIM_Base_InitTypeDef::ClockDivision*
Specifies the clock division. This parameter can be a value of [TIM_ClockDivision](#)

41.2.2 TIM_OC_InitTypeDef

Data Fields

- *uint32_t OCMODE*
- *uint32_t Pulse*
- *uint32_t OCPolarity*
- *uint32_t OCFastMode*
- *uint32_t OCIdleState*

Field Documentation

- *uint32_t TIM_OC_InitTypeDef::OCMode*
Specifies the TIM mode. This parameter can be a value of [TIM_Output_Compare_and_PWM_modes](#)
- *uint32_t TIM_OC_InitTypeDef::Pulse*
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF

- ***uint32_t TIM_OC_InitTypeDef::OCpolarity***
Specifies the output polarity. This parameter can be a value of [TIM_Output_Compare_Polarity](#)
- ***uint32_t TIM_OC_InitTypeDef::OCFastMode***
Specifies the Fast mode state. This parameter can be a value of [TIM_Output_Fast_State](#)
Note: This parameter is valid only in PWM1 and PWM2 mode.
- ***uint32_t TIM_OC_InitTypeDef::OCIdleState***
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of [TIM_Output_Compare_Idle_State](#).

41.2.3 TIM_OnePulse_InitTypeDef

Data Fields

- ***uint32_t OCMODE***
- ***uint32_t Pulse***
- ***uint32_t OCPolarity***
- ***uint32_t OCIdleState***
- ***uint32_t ICPolarity***
- ***uint32_t ICSelection***
- ***uint32_t ICFILTER***

Field Documentation

- ***uint32_t TIM_OnePulse_InitTypeDef::OCMode***
Specifies the TIM mode. This parameter can be a value of [TIM_Output_Compare_and_PWM_modes](#)
- ***uint32_t TIM_OnePulse_InitTypeDef::Pulse***
Specifies the pulse value to be loaded into the Capture Compare Register. This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF
- ***uint32_t TIM_OnePulse_InitTypeDef::OCpolarity***
Specifies the output polarity. This parameter can be a value of [TIM_Output_Compare_Polarity](#)
- ***uint32_t TIM_OnePulse_InitTypeDef::OCIdleState***
Specifies the TIM Output Compare pin state during Idle state. This parameter can be a value of [TIM_Output_Compare_Idle_State](#).
- ***uint32_t TIM_OnePulse_InitTypeDef::ICPolarity***
Specifies the active edge of the input signal. This parameter can be a value of [TIM_Input_Capture_Polarity](#)
- ***uint32_t TIM_OnePulse_InitTypeDef::ICSelection***
Specifies the input. This parameter can be a value of [TIM_Input_Capture_Selection](#)
- ***uint32_t TIM_OnePulse_InitTypeDef::ICFilter***
Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

41.2.4 TIM_IC_InitTypeDef

Data Fields

- *uint32_t ICPolarity*
- *uint32_t ICSelection*
- *uint32_t ICPrescaler*
- *uint32_t ICFilter*

Field Documentation

- *uint32_t TIM_IC_InitTypeDef::ICPolarity*
Specifies the active edge of the input signal. This parameter can be a value of [TIM_Input_Capture_Polarity](#)
- *uint32_t TIM_IC_InitTypeDef::ICSelection*
Specifies the input. This parameter can be a value of [TIM_Input_Capture_Selection](#)
- *uint32_t TIM_IC_InitTypeDef::ICPrescaler*
Specifies the Input Capture Prescaler. This parameter can be a value of [TIM_Input_Capture_Prescaler](#)
- *uint32_t TIM_IC_InitTypeDef::ICFilter*
Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

41.2.5 TIM_Encoder_InitTypeDef

Data Fields

- *uint32_t EncoderMode*
- *uint32_t IC1Polarity*
- *uint32_t IC1Selection*
- *uint32_t IC1Prescaler*
- *uint32_t IC1Filter*
- *uint32_t IC2Polarity*
- *uint32_t IC2Selection*
- *uint32_t IC2Prescaler*
- *uint32_t IC2Filter*

Field Documentation

- *uint32_t TIM_Encoder_InitTypeDef::EncoderMode*
Specifies the active edge of the input signal. This parameter can be a value of [TIM_Encoder_Mode](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC1Polarity*
Specifies the active edge of the input signal. This parameter can be a value of [TIM_Input_Capture_Polarity](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC1Selection*
Specifies the input. This parameter can be a value of [TIM_Input_Capture_Selection](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC1Prescaler*
Specifies the Input Capture Prescaler. This parameter can be a value of [TIM_Input_Capture_Prescaler](#)
- *uint32_t TIM_Encoder_InitTypeDef::IC1Filter*
Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

- ***uint32_t TIM_Encoder_InitTypeDef::IC2Polarity***
Specifies the active edge of the input signal. This parameter can be a value of [TIM_Input_Capture_Polarity](#)
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Selection***
Specifies the input. This parameter can be a value of [TIM_Input_Capture_Selection](#)
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Prescaler***
Specifies the Input Capture Prescaler. This parameter can be a value of [TIM_Input_Capture_Prescaler](#)
- ***uint32_t TIM_Encoder_InitTypeDef::IC2Filter***
Specifies the input capture filter. This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

41.2.6 TIM_ClockConfigTypeDef

Data Fields

- ***uint32_t ClockSource***
- ***uint32_t ClockPolarity***
- ***uint32_t ClockPrescaler***
- ***uint32_t ClockFilter***

Field Documentation

- ***uint32_t TIM_ClockConfigTypeDef::ClockSource***
TIM clock sources This parameter can be a value of [TIM_Clock_Source](#)
- ***uint32_t TIM_ClockConfigTypeDef::ClockPolarity***
TIM clock polarity This parameter can be a value of [TIM_Clock_Polarity](#)
- ***uint32_t TIM_ClockConfigTypeDef::ClockPrescaler***
TIM clock prescaler This parameter can be a value of [TIM_Clock_Prescaler](#)
- ***uint32_t TIM_ClockConfigTypeDef::ClockFilter***
TIM clock filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

41.2.7 TIM_ClearInputConfigTypeDef

Data Fields

- ***uint32_t ClearInputState***
- ***uint32_t ClearInputSource***
- ***uint32_t ClearInputPolarity***
- ***uint32_t ClearInputPrescaler***
- ***uint32_t ClearInputFilter***

Field Documentation

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputState***
TIM clear Input state This parameter can be ENABLE or DISABLE
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputSource***
TIM clear Input sources This parameter can be a value of [TIM_ClearInput_Source](#)

- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPolarity***
TIM Clear Input polarity This parameter can be a value of [TIM_ClearInput_Polarity](#)
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputPrescaler***
TIM Clear Input prescaler This parameter can be a value of [TIM_ClearInput_Prescaler](#)
- ***uint32_t TIM_ClearInputConfigTypeDef::ClearInputFilter***
TIM Clear Input filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

41.2.8 TIM_SlaveConfigTypeDef

Data Fields

- ***uint32_t SlaveMode***
- ***uint32_t InputTrigger***
- ***uint32_t TriggerPolarity***
- ***uint32_t TriggerPrescaler***
- ***uint32_t TriggerFilter***

Field Documentation

- ***uint32_t TIM_SlaveConfigTypeDef::SlaveMode***
Slave mode selection This parameter can be a value of [TIM_Slave_Mode](#)
- ***uint32_t TIM_SlaveConfigTypeDef::InputTrigger***
Input Trigger source This parameter can be a value of [TIM_Trigger_Selection](#)
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPolarity***
Input Trigger polarity This parameter can be a value of [TIM_Trigger_Polarity](#)
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerPrescaler***
Input trigger prescaler This parameter can be a value of [TIM_Trigger_Prescaler](#)
- ***uint32_t TIM_SlaveConfigTypeDef::TriggerFilter***
Input trigger filter This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF

41.2.9 TIM_HandleTypeDef

Data Fields

- ***TIM_TypeDef * Instance***
- ***TIM_Base_InitTypeDef Init***
- ***HAL_TIM_ActiveChannel Channel***
- ***DMA_HandleTypeDef * hdma***
- ***HAL_LockTypeDef Lock***
- ***__IO HAL_TIM_StateTypeDef State***

Field Documentation

- ***TIM_TypeDef* TIM_HandleTypeDef::Instance***
Register base address

- ***TIM_Base_InitTypeDef TIM_HandleTypeDef::Init***
TIM Time Base required parameters
- ***HAL_TIM_ActiveChannel TIM_HandleTypeDef::Channel***
Active channel
- ***DMA_HandleTypeDef* TIM_HandleTypeDef::hdma[7]***
DMA Handlers array This array is accessed by a [TIM_DMA_Handle_index](#)
- ***HAL_LockTypeDef TIM_HandleTypeDef::Lock***
Locking object
- ***__IO HAL_TIM_StateTypeDef TIM_HandleTypeDef::State***
TIM operation state

41.3 TIM Firmware driver API description

41.3.1 TIMER Generic features

The Timer features include:

1. 16-bit up, down, up/down auto-reload counter.
2. 16-bit programmable prescaler allowing dividing (also on the fly) the counter clock frequency either by any factor between 1 and 65536.
3. Up to 4 independent channels for:
 - Input Capture
 - Output Compare
 - PWM generation (Edge and Center-aligned Mode)
 - One-pulse mode output
4. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
5. Supports incremental (quadrature) encoder

41.3.2 How to use this driver

1. Initialize the TIM low level resources by implementing the following functions depending from feature used :
 - Time Base : `HAL_TIM_Base_MspInit()`
 - Input Capture : `HAL_TIM_IC_MspInit()`
 - Output Compare : `HAL_TIM_OC_MspInit()`
 - PWM generation : `HAL_TIM_PWM_MspInit()`
 - One-pulse mode output : `HAL_TIM_OnePulse_MspInit()`
 - Encoder mode output : `HAL_TIM_Encoder_MspInit()`
2. Initialize the TIM low level resources :
 - a. Enable the TIM interface clock using `__HAL_RCC_TIMx_CLK_ENABLE()`;
 - b. TIM pins configuration
 - Enable the clock for the TIM GPIOs using the following function:
`__HAL_RCC_GPIOx_CLK_ENABLE()`;
 - Configure these TIM pins in Alternate function mode using
`HAL_GPIO_Init()`;
3. The external Clock can be configured, if needed (the default clock is the internal clock from the APBx), using the following function: `HAL_TIM_ConfigClockSource`, the clock configuration should be done before any start function.
4. Configure the TIM in the desired functioning mode using one of the Initialization function of this driver:
 - `HAL_TIM_Base_Init`: to use the Timer to generate a simple time base

- HAL_TIM_OC_Init and HAL_TIM_OC_ConfigChannel: to use the Timer to generate an Output Compare signal.
 - HAL_TIM_PWM_Init and HAL_TIM_PWM_ConfigChannel: to use the Timer to generate a PWM signal.
 - HAL_TIM_IC_Init and HAL_TIM_IC_ConfigChannel: to use the Timer to measure an external signal.
 - HAL_TIM_OnePulse_Init and HAL_TIM_OnePulse_ConfigChannel: to use the Timer in One Pulse Mode.
 - HAL_TIM_Encoder_Init: to use the Timer Encoder Interface.
5. Activate the TIM peripheral using one of the start functions depending from the feature used:
- Time Base : HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_Base_Start_IT()
 - Input Capture : HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_IC_Start_IT()
 - Output Compare : HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_OC_Start_IT()
 - PWM generation : HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_TIM_PWM_Start_IT()
 - One-pulse mode output : HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_Start_IT()
 - Encoder mode output : HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_DMA(), HAL_TIM_Encoder_Start_IT().
6. The DMA Burst is managed with the two following functions:
HAL_TIM_DMABurst_WriteStart() HAL_TIM_DMABurst_ReadStart()

41.3.3 Time Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Time Base.
- Stop the Time Base.
- Start the Time Base and enable interrupt.
- Stop the Time Base and disable interrupt.
- Start the Time Base and enable DMA transfer.
- Stop the Time Base and disable DMA transfer.

This section contains the following APIs:

- [*HAL_TIM_Base_Init\(\)*](#)
- [*HAL_TIM_Base_DeInit\(\)*](#)
- [*HAL_TIM_Base_MspInit\(\)*](#)
- [*HAL_TIM_Base_MspDeInit\(\)*](#)
- [*HAL_TIM_Base_Start\(\)*](#)
- [*HAL_TIM_Base_Stop\(\)*](#)
- [*HAL_TIM_Base_Start_IT\(\)*](#)
- [*HAL_TIM_Base_Stop_IT\(\)*](#)
- [*HAL_TIM_Base_Start_DMA\(\)*](#)
- [*HAL_TIM_Base_Stop_DMA\(\)*](#)

41.3.4 Time Output Compare functions

This section provides functions allowing to:

- Initialize and configure the TIM Output Compare.

- De-initialize the TIM Output Compare.
- Start the Time Output Compare.
- Stop the Time Output Compare.
- Start the Time Output Compare and enable interrupt.
- Stop the Time Output Compare and disable interrupt.
- Start the Time Output Compare and enable DMA transfer.
- Stop the Time Output Compare and disable DMA transfer.

This section contains the following APIs:

- [*HAL_TIM_OC_Init\(\)*](#)
- [*HAL_TIM_OC_DeInit\(\)*](#)
- [*HAL_TIM_OC_MspInit\(\)*](#)
- [*HAL_TIM_OC_MspDeInit\(\)*](#)
- [*HAL_TIM_OC_Start\(\)*](#)
- [*HAL_TIM_OC_Stop\(\)*](#)
- [*HAL_TIM_OC_Start_IT\(\)*](#)
- [*HAL_TIM_OC_Stop_IT\(\)*](#)
- [*HAL_TIM_OC_Start_DMA\(\)*](#)
- [*HAL_TIM_OC_Stop_DMA\(\)*](#)

41.3.5 Time PWM functions

This section provides functions allowing to:

- Initialize and configure the TIM PWM.
- De-initialize the TIM PWM.
- Start the Time PWM.
- Stop the Time PWM.
- Start the Time PWM and enable interrupt.
- Stop the Time PWM and disable interrupt.
- Start the Time PWM and enable DMA transfer.
- Stop the Time PWM and disable DMA transfer.

This section contains the following APIs:

- [*HAL_TIM_PWM_Init\(\)*](#)
- [*HAL_TIM_PWM_DeInit\(\)*](#)
- [*HAL_TIM_PWM_MspInit\(\)*](#)
- [*HAL_TIM_PWM_MspDeInit\(\)*](#)
- [*HAL_TIM_PWM_Start\(\)*](#)
- [*HAL_TIM_PWM_Stop\(\)*](#)
- [*HAL_TIM_PWM_Start_IT\(\)*](#)
- [*HAL_TIM_PWM_Stop_IT\(\)*](#)
- [*HAL_TIM_PWM_Start_DMA\(\)*](#)
- [*HAL_TIM_PWM_Stop_DMA\(\)*](#)

41.3.6 Time Input Capture functions

This section provides functions allowing to:

- Initialize and configure the TIM Input Capture.
- De-initialize the TIM Input Capture.
- Start the Time Input Capture.
- Stop the Time Input Capture.
- Start the Time Input Capture and enable interrupt.
- Stop the Time Input Capture and disable interrupt.

- Start the Time Input Capture and enable DMA transfer.
- Stop the Time Input Capture and disable DMA transfer.

This section contains the following APIs:

- [*HAL_TIM_IC_Init\(\)*](#)
- [*HAL_TIM_IC_DeInit\(\)*](#)
- [*HAL_TIM_IC_MspInit\(\)*](#)
- [*HAL_TIM_IC_MspDeInit\(\)*](#)
- [*HAL_TIM_IC_Start\(\)*](#)
- [*HAL_TIM_IC_Stop\(\)*](#)
- [*HAL_TIM_IC_Start_IT\(\)*](#)
- [*HAL_TIM_IC_Stop_IT\(\)*](#)
- [*HAL_TIM_IC_Start_DMA\(\)*](#)
- [*HAL_TIM_IC_Stop_DMA\(\)*](#)

41.3.7 Time One Pulse functions

This section provides functions allowing to:

- Initialize and configure the TIM One Pulse.
- De-initialize the TIM One Pulse.
- Start the Time One Pulse.
- Stop the Time One Pulse.
- Start the Time One Pulse and enable interrupt.
- Stop the Time One Pulse and disable interrupt.
- Start the Time One Pulse and enable DMA transfer.
- Stop the Time One Pulse and disable DMA transfer.

This section contains the following APIs:

- [*HAL_TIM_OnePulse_Init\(\)*](#)
- [*HAL_TIM_OnePulse_DeInit\(\)*](#)
- [*HAL_TIM_OnePulse_MspInit\(\)*](#)
- [*HAL_TIM_OnePulse_MspDeInit\(\)*](#)
- [*HAL_TIM_OnePulse_Start\(\)*](#)
- [*HAL_TIM_OnePulse_Stop\(\)*](#)
- [*HAL_TIM_OnePulse_Start_IT\(\)*](#)
- [*HAL_TIM_OnePulse_Stop_IT\(\)*](#)

41.3.8 Time Encoder functions

This section provides functions allowing to:

- Initialize and configure the TIM Encoder.
- De-initialize the TIM Encoder.
- Start the Time Encoder.
- Stop the Time Encoder.
- Start the Time Encoder and enable interrupt.
- Stop the Time Encoder and disable interrupt.
- Start the Time Encoder and enable DMA transfer.
- Stop the Time Encoder and disable DMA transfer.

This section contains the following APIs:

- [*HAL_TIM_Encoder_Init\(\)*](#)
- [*HAL_TIM_Encoder_DeInit\(\)*](#)
- [*HAL_TIM_Encoder_MspInit\(\)*](#)

- [*HAL_TIM_Encoder_MspDeInit\(\)*](#)
- [*HAL_TIM_Encoder_Start\(\)*](#)
- [*HAL_TIM_Encoder_Stop\(\)*](#)
- [*HAL_TIM_Encoder_Start_IT\(\)*](#)
- [*HAL_TIM_Encoder_Stop_IT\(\)*](#)
- [*HAL_TIM_Encoder_Start_DMA\(\)*](#)
- [*HAL_TIM_Encoder_Stop_DMA\(\)*](#)

41.3.9 IRQ handler management

This section provides Timer IRQ handler function.

This section contains the following APIs:

- [*HAL_TIM_IRQHandler\(\)*](#)

41.3.10 Peripheral Control functions

This section provides functions allowing to:

- Configure The Input Output channels for OC, PWM, IC or One Pulse mode.
- Configure External Clock source.
- Configure Complementary channels, break features and dead time.
- Configure Master and the Slave synchronization.
- Configure the DMA Burst Mode.

This section contains the following APIs:

- [*HAL_TIM_OC_ConfigChannel\(\)*](#)
- [*HAL_TIM_IC_ConfigChannel\(\)*](#)
- [*HAL_TIM_PWM_ConfigChannel\(\)*](#)
- [*HAL_TIM_OnePulse_ConfigChannel\(\)*](#)
- [*HAL_TIM_DMABurst_WriteStart\(\)*](#)
- [*HAL_TIM_DMABurst_WriteStop\(\)*](#)
- [*HAL_TIM_DMABurst_ReadStart\(\)*](#)
- [*HAL_TIM_DMABurst_ReadStop\(\)*](#)
- [*HAL_TIM_GenerateEvent\(\)*](#)
- [*HAL_TIM_ConfigOCrefClear\(\)*](#)
- [*HAL_TIM_ConfigClockSource\(\)*](#)
- [*HAL_TIM_ConfigTI1Input\(\)*](#)
- [*HAL_TIM_SlaveConfigSynchronization\(\)*](#)
- [*HAL_TIM_SlaveConfigSynchronization_IT\(\)*](#)
- [*HAL_TIM_ReadCapturedValue\(\)*](#)

41.3.11 TIM Callbacks functions

This section provides TIM callback functions:

- Timer Period elapsed callback
- Timer Output Compare callback
- Timer Input capture callback
- Timer Trigger callback
- Timer Error callback

This section contains the following APIs:

- [*HAL_TIM_PeriodElapsedCallback\(\)*](#)
- [*HAL_TIM_OC_DelayElapsedCallback\(\)*](#)

- [HAL_TIM_IC_CaptureCallback\(\)](#)
- [HAL_TIM_PWM_PulseFinishedCallback\(\)](#)
- [HAL_TIM_TriggerCallback\(\)](#)
- [HAL_TIM_ErrorCallback\(\)](#)

41.3.12 Peripheral State functions

This subsection permit to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- [HAL_TIM_Base_GetState\(\)](#)
- [HAL_TIM_OC_GetState\(\)](#)
- [HAL_TIM_PWM_GetState\(\)](#)
- [HAL_TIM_IC_GetState\(\)](#)
- [HAL_TIM_OnePulse_GetState\(\)](#)
- [HAL_TIM_Encoder_GetState\(\)](#)
- [TIM_DMAError\(\)](#)
- [TIM_DMADelayPulseCplt\(\)](#)
- [TIM_DMACaptureCplt\(\)](#)

41.3.13 HAL_TIM_Base_Init

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Time base Unit according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim: TIM Base handle
Return values	<ul style="list-style-type: none"> • HAL status

41.3.14 HAL_TIM_Base_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM Base peripheral.
Parameters	<ul style="list-style-type: none"> • htim: TIM Base handle
Return values	<ul style="list-style-type: none"> • HAL status

41.3.15 HAL_TIM_Base_MspInit

Function Name	void HAL_TIM_Base_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Base MSP.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle
Return values	<ul style="list-style-type: none"> • None

41.3.16 HAL_TIM_Base_MspDeInit

Function Name	void HAL_TIM_Base_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Base MSP.

- | | |
|---------------|---|
| Parameters | <ul style="list-style-type: none">• htim: TIM handle |
| Return values | <ul style="list-style-type: none">• None |

41.3.17 HAL_TIM_Base_Start

- | | |
|----------------------|---|
| Function Name | HAL_StatusTypeDef HAL_TIM_Base_Start
(TIM_HandleTypeDef * htim) |
| Function Description | Starts the TIM Base generation. |
| Parameters | <ul style="list-style-type: none">• htim: : TIM handle |
| Return values | <ul style="list-style-type: none">• HAL status |

41.3.18 HAL_TIM_Base_Stop

- | | |
|----------------------|---|
| Function Name | HAL_StatusTypeDef HAL_TIM_Base_Stop
(TIM_HandleTypeDef * htim) |
| Function Description | Stops the TIM Base generation. |
| Parameters | <ul style="list-style-type: none">• htim: : TIM handle |
| Return values | <ul style="list-style-type: none">• HAL status |

41.3.19 HAL_TIM_Base_Start_IT

- | | |
|----------------------|--|
| Function Name | HAL_StatusTypeDef HAL_TIM_Base_Start_IT
(TIM_HandleTypeDef * htim) |
| Function Description | Starts the TIM Base generation in interrupt mode. |
| Parameters | <ul style="list-style-type: none">• htim: : TIM handle |
| Return values | <ul style="list-style-type: none">• HAL status |

41.3.20 HAL_TIM_Base_Stop_IT

- | | |
|----------------------|---|
| Function Name | HAL_StatusTypeDef HAL_TIM_Base_Stop_IT
(TIM_HandleTypeDef * htim) |
| Function Description | Stops the TIM Base generation in interrupt mode. |
| Parameters | <ul style="list-style-type: none">• htim: : TIM handle |
| Return values | <ul style="list-style-type: none">• HAL status |

41.3.21 HAL_TIM_Base_Start_DMA

- | | |
|----------------------|---|
| Function Name | HAL_StatusTypeDef HAL_TIM_Base_Start_DMA
(TIM_HandleTypeDef * htim, uint32_t * pData, uint16_t Length) |
| Function Description | Starts the TIM Base generation in DMA mode. |
| Parameters | <ul style="list-style-type: none">• htim: : TIM handle• pData: The source Buffer address.• Length: The length of data to be transferred from memory to peripheral. |
| Return values | <ul style="list-style-type: none">• HAL status |

41.3.22 HAL_TIM_Base_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_Base_Stop_DMA (TIM_HandleTypeDef * htim)
Function Description	Stops the TIM Base generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle
Return values	<ul style="list-style-type: none"> • HAL status

41.3.23 HAL_TIM_OC_Init

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Output Compare according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim: TIM Output Compare handle
Return values	<ul style="list-style-type: none"> • HAL status

41.3.24 HAL_TIM_OC_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> • htim: TIM Output Compare handle
Return values	<ul style="list-style-type: none"> • HAL status

41.3.25 HAL_TIM_OC_MspInit

Function Name	void HAL_TIM_OC_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle
Return values	<ul style="list-style-type: none"> • None

41.3.26 HAL_TIM_OC_MspDeInit

Function Name	void HAL_TIM_OC_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Output Compare MSP.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle
Return values	<ul style="list-style-type: none"> • None

41.3.27 HAL_TIM_OC_Start

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Output Compare signal generation.

Parameters	<ul style="list-style-type: none"> • htim: : TIM Output Compare handle • Channel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.28 HAL_TIM_OC_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.29 HAL_TIM_OC_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Output Compare signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM OC handle • Channel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.30 HAL_TIM_OC_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Output Compare handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.31 HAL_TIM_OC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM Output Compare signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Output Compare handle • Channel: : TIM Channel to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected • pData: The source Buffer address. • Length: The length of data to be transferred from memory to TIM peripheral
Return values	<ul style="list-style-type: none"> • HAL status

41.3.32 HAL_TIM_OC_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_OC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Output Compare signal generation in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Output Compare handle • Channel: : TIM Channel to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.33 HAL_TIM_PWM_Init

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM PWM Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle
Return values	<ul style="list-style-type: none"> • HAL status

41.3.34 HAL_TIM_PWM_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle
Return values	<ul style="list-style-type: none"> • HAL status

41.3.35 HAL_TIM_PWM_MspInit

Function Name	void HAL_TIM_PWM_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM PWM MSP.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None

41.3.36 HAL_TIM_PWM_MspDeInit

Function Name	void HAL_TIM_PWM_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM PWM MSP.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None

41.3.37 HAL_TIM_PWM_Start

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle• Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none">• HAL status

41.3.38 HAL_TIM_PWM_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the PWM signal generation.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle• Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none">• HAL status

41.3.39 HAL_TIM_PWM_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the PWM signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none">• htim: : TIM handle• Channel: : TIM Channel to be disabled This parameter can

be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected
TIM_CHANNEL_2: TIM Channel 2 selected
TIM_CHANNEL_3: TIM Channel 3 selected
TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

41.3.40 HAL_TIM_PWM_Stop_IT

Function Name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_IT
(TIM_HandleTypeDef * htim, uint32_t Channel)

Function Description

Stops the PWM signal generation in interrupt mode.

Parameters

- **htim:** : TIM handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected
TIM_CHANNEL_2: TIM Channel 2 selected
TIM_CHANNEL_3: TIM Channel 3 selected
TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

41.3.41 HAL_TIM_PWM_Start_DMA

Function Name

HAL_StatusTypeDef HAL_TIM_PWM_Start_DMA
(TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

Function Description

Starts the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** : TIM handle
- **Channel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected
TIM_CHANNEL_2: TIM Channel 2 selected
TIM_CHANNEL_3: TIM Channel 3 selected
TIM_CHANNEL_4: TIM Channel 4 selected
- **pData:** The source Buffer address.
- **Length:** The length of data to be transferred from memory to TIM peripheral

Return values

- HAL status

41.3.42 HAL_TIM_PWM_Stop_DMA

Function Name

HAL_StatusTypeDef HAL_TIM_PWM_Stop_DMA
(TIM_HandleTypeDef * htim, uint32_t Channel)

Function Description

Stops the TIM PWM signal generation in DMA mode.

Parameters

- **htim:** : TIM handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selected
TIM_CHANNEL_2: TIM Channel 2 selected
TIM_CHANNEL_3: TIM Channel 3 selected
TIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

41.3.43 HAL_TIM_IC_Init

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Init (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Input Capture Time base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none">• htim: TIM Input Capture handle
Return values	<ul style="list-style-type: none">• HAL status

41.3.44 HAL_TIM_IC_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_IC_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM peripheral.
Parameters	<ul style="list-style-type: none">• htim: TIM Input Capture handle
Return values	<ul style="list-style-type: none">• HAL status

41.3.45 HAL_TIM_IC_MspInit

Function Name	void HAL_TIM_IC_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None

41.3.46 HAL_TIM_IC_MspDeInit

Function Name	void HAL_TIM_IC_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Input Capture MSP.
Parameters	<ul style="list-style-type: none">• htim: TIM handle
Return values	<ul style="list-style-type: none">• None

41.3.47 HAL_TIM_IC_Start

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none">• htim: : TIM Input Capture handle• Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none">• HAL status

41.3.48 HAL_TIM_IC_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Input Capture measurement.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.49 HAL_TIM_IC_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Input Capture handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.50 HAL_TIM_IC_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Input Capture measurement in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.51 HAL_TIM_IC_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
Function Description	Starts the TIM Input Capture measurement in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Input Capture handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

- **pData:** The destination Buffer address.
- **Length:** The length of data to be transferred from TIM peripheral to memory.

Return values

- HAL status

41.3.52 HAL_TIM_IC_Stop_DMA

Function Name **HAL_StatusTypeDef HAL_TIM_IC_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function Description Stops the TIM Input Capture measurement in DMA mode.

Parameters

- **htim:** : TIM Input Capture handle
- **Channel:** : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

41.3.53 HAL_TIM_OnePulse_Init

Function Name **HAL_StatusTypeDef HAL_TIM_OnePulse_Init (TIM_HandleTypeDef * htim, uint32_t OnePulseMode)**

Function Description Initializes the TIM One Pulse Time Base according to the specified parameters in the TIM_HandleTypeDef and create the associated handle.

Parameters

- **htim:** TIM OnePulse handle
- **OnePulseMode:** Select the One pulse mode. This parameter can be one of the following values: TIM_OPMODE_SINGLE: Only one pulse will be generated.TIM_OPMODE_REPETITIVE: Repetitive pulses will be generated.

Return values

- HAL status

41.3.54 HAL_TIM_OnePulse_DeInit

Function Name **HAL_StatusTypeDef HAL_TIM_OnePulse_DeInit (TIM_HandleTypeDef * htim)**

Function Description DeInitializes the TIM One Pulse.

Parameters

- **htim:** TIM One Pulse handle

Return values

- HAL status

41.3.55 HAL_TIM_OnePulse_MspInit

Function Name **void HAL_TIM_OnePulse_MspInit (TIM_HandleTypeDef * htim)**

Function Description Initializes the TIM One Pulse MSP.

Parameters

- **htim:** TIM handle

Return values

- None

41.3.56 HAL_TIM_OnePulse_MspDeInit

Function Name	void HAL_TIM_OnePulse_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM One Pulse MSP.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle
Return values	<ul style="list-style-type: none"> • None

41.3.57 HAL_TIM_OnePulse_Start

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Start (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Starts the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> • htim: : TIM One Pulse handle • OutputChannel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.58 HAL_TIM_OnePulse_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Stop (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Stops the TIM One Pulse signal generation.
Parameters	<ul style="list-style-type: none"> • htim: : TIM One Pulse handle • OutputChannel: : TIM Channels to be disable This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.59 HAL_TIM_OnePulse_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Start_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
Function Description	Starts the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM One Pulse handle • OutputChannel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.60 HAL_TIM_OnePulse_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_OnePulse_Stop_IT (TIM_HandleTypeDef * htim, uint32_t OutputChannel)
---------------	--

Function Description	Stops the TIM One Pulse signal generation in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM One Pulse handle • OutputChannel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.61 HAL_TIM_Encoder_Init

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Init (TIM_HandleTypeDef * htim, TIM_Encoder_InitTypeDef * sConfig)
Function Description	Initializes the TIM Encoder Interface and create the associated handle.
Parameters	<ul style="list-style-type: none"> • htim: TIM Encoder Interface handle • sConfig: TIM Encoder Interface configuration structure
Return values	<ul style="list-style-type: none"> • HAL status

41.3.62 HAL_TIM_Encoder_DeInit

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_DeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes the TIM Encoder interface.
Parameters	<ul style="list-style-type: none"> • htim: TIM Encoder handle
Return values	<ul style="list-style-type: none"> • HAL status

41.3.63 HAL_TIM_Encoder_MspInit

Function Name	void HAL_TIM_Encoder_MspInit (TIM_HandleTypeDef * htim)
Function Description	Initializes the TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle
Return values	<ul style="list-style-type: none"> • None

41.3.64 HAL_TIM_Encoder_MspDeInit

Function Name	void HAL_TIM_Encoder_MspDeInit (TIM_HandleTypeDef * htim)
Function Description	DeInitializes TIM Encoder Interface MSP.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle
Return values	<ul style="list-style-type: none"> • None

41.3.65 HAL_TIM_Encoder_Start

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
---------------	--

Function Description	Starts the TIM Encoder Interface.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.66 HAL_TIM_Encoder_Stop

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Encoder Interface.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.67 HAL_TIM_Encoder_Start_IT

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Starts the TIM Encoder Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.68 HAL_TIM_Encoder_Stop_IT

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Encoder Interface in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be disabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.69 HAL_TIM_Encoder_Start_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData1, uint32_t * pData2, uint16_t Length)
Function Description	Starts the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected • pData1: The destination Buffer address for IC1. • pData2: The destination Buffer address for IC2. • Length: The length of data to be transferred from TIM peripheral to memory.
Return values	<ul style="list-style-type: none"> • HAL status

41.3.70 HAL_TIM_Encoder_Stop_DMA

Function Name	HAL_StatusTypeDef HAL_TIM_Encoder_Stop_DMA (TIM_HandleTypeDef * htim, uint32_t Channel)
Function Description	Stops the TIM Encoder Interface in DMA mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM Encoder Interface handle • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_ALL: TIM Channel 1 and TIM Channel 2 are selected
Return values	<ul style="list-style-type: none"> • HAL status

41.3.71 HAL_TIM_IRQHandler

Function Name	void HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)
Function Description	This function handles TIM interrupts requests.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle
Return values	<ul style="list-style-type: none"> • None

41.3.72 HAL_TIM_OC_ConfigChannel

Function Name	HAL_StatusTypeDef HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)
Function Description	Initializes the TIM Output Compare Channels according to the specified parameters in the TIM_OC_InitTypeDef.
Parameters	<ul style="list-style-type: none"> • htim: TIM Output Compare handle • sConfig: TIM Output Compare configuration structure • Channel: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM

Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2
 selectedTIM_CHANNEL_3: TIM Channel 3
 selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

41.3.73 HAL_TIM_IC_ConfigChannel

Function Name `HAL_StatusTypeDef HAL_TIM_IC_ConfigChannel(TIM_HandleTypeDef * htim, TIM_IC_InitTypeDef * sConfig, uint32_t Channel)`

Function Description Initializes the TIM Input Capture Channels according to the specified parameters in the TIM_IC_InitTypeDef.

Parameters

- **htim**: TIM IC handle
- **sConfig**: TIM Input Capture configuration structure
- **Channel**: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

41.3.74 HAL_TIM_PWM_ConfigChannel

Function Name `HAL_StatusTypeDef HAL_TIM_PWM_ConfigChannel(TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)`

Function Description Initializes the TIM PWM channels according to the specified parameters in the TIM_OC_InitTypeDef.

Parameters

- **htim**: TIM handle
- **sConfig**: TIM PWM configuration structure
- **Channel**: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values

- HAL status

41.3.75 HAL_TIM_OnePulse_ConfigChannel

Function Name `HAL_StatusTypeDef HAL_TIM_OnePulse_ConfigChannel(TIM_HandleTypeDef * htim, TIM_OnePulse_InitTypeDef * sConfig, uint32_t OutputChannel, uint32_t InputChannel)`

Function Description Initializes the TIM One Pulse Channels according to the specified parameters in the TIM_OnePulse_InitTypeDef.

Parameters

- **htim**: TIM One Pulse handle
- **sConfig**: TIM One Pulse configuration structure
- **OutputChannel**: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected

- **InputChannel:** : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selected

Return values

- HAL status

41.3.76 HAL_TIM_DMABurst_WriteStart

Function Name **HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStart (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)**

Function Description Configure the DMA Burst to transfer Data from the memory to the TIM peripheral.

- Parameters**
- **htim:** TIM handle
 - **BurstBaseAddress:** : TIM Base address from where the DMA will start the Data write This parameter can be one of the following values:
TIM_DMABASE_CR1TIM_DMABASE_CR2TIM_DMABASE_SMCRTIM_DMABASE_DIERTIM_DMABASE_SRTIM_DMABASE_EGRTIM_DMABASE_CCMR1TIM_DMABASE_CCMR2TIM_DMABASE_CCERTIM_DMABASE_CNTTIM_DMABASE_PSCTIM_DMABASE_ARRTIM_DMABASE_CCR1TIM_DMABASE_CCR2TIM_DMABASE_CCR3TIM_DMABASE_CCR4TIM_DMABASE_DCR
 - **BurstRequestSrc:** TIM DMA Request sources This parameter can be one of the following values: TIM_DMA_UPDATE: TIM update Interrupt sourceTIM_DMA_CC1: TIM Capture Compare 1 DMA sourceTIM_DMA_CC2: TIM Capture Compare 2 DMA sourceTIM_DMA_CC3: TIM Capture Compare 3 DMA sourceTIM_DMA_CC4: TIM Capture Compare 4 DMA sourceTIM_DMA_TRIGGER: TIM Trigger DMA source
 - **BurstBuffer:** The Buffer address.
 - **BurstLength:** DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.
- Return values**
- HAL status

41.3.77 HAL_TIM_DMABurst_WriteStop

Function Name	HAL_StatusTypeDef HAL_TIM_DMABurst_WriteStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)
Function Description	Stops the TIM DMA Burst mode.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle • BurstRequestSrc: TIM DMA Request sources to disable
Return values	<ul style="list-style-type: none"> • HAL status

41.3.78 HAL_TIM_DMABurst_ReadStart

Function Name **HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStart** (TIM_HandleTypeDef * htim, uint32_t BurstBaseAddress, uint32_t BurstRequestSrc, uint32_t * BurstBuffer, uint32_t BurstLength)

Function Description Configure the DMA Burst to transfer Data from the TIM peripheral to the memory.

- Parameters**
- **htim**: TIM handle
 - **BurstBaseAddress**: : TIM Base address from where the DMA will starts the Data read This parameter can be one of the following values:
TIM_DMABASE_CR1TIM_DMABASE_CR2TIM_DMABASE_SMCRTIM_DMABASE_DIERTIM_DMABASE_SRTIM_DMABASE_EGRTIM_DMABASE_CCMR1TIM_DMABASE_CCMR2TIM_DMABASE_CCERTIM_DMABASE_CNTTIM_DMABASE_PSCTIM_DMABASE_ARRTIM_DMABASE_CCR1TIM_DMABASE_CCR2TIM_DMABASE_CCR3TIM_DMABASE_CCR4TIM_DMABASE_DCR
 - **BurstRequestSrc**: TIM DMA Request sources This parameter can be one of the following values: TIM_DMA_UPDATE: TIM update Interrupt sourceTIM_DMA_CC1: TIM Capture Compare 1 DMA sourceTIM_DMA_CC2: TIM Capture Compare 2 DMA sourceTIM_DMA_CC3: TIM Capture Compare 3 DMA sourceTIM_DMA_CC4: TIM Capture Compare 4 DMA sourceTIM_DMA_TRIGGER: TIM Trigger DMA source
 - **BurstBuffer**: The Buffer address.
 - **BurstLength**: DMA Burst length. This parameter can be one value between: TIM_DMABURSTLENGTH_1TRANSFER and TIM_DMABURSTLENGTH_18TRANSFERS.
- Return values**
- HAL status

41.3.79 HAL_TIM_DMABurst_ReadStop

Function Name	HAL_StatusTypeDef HAL_TIM_DMABurst_ReadStop (TIM_HandleTypeDef * htim, uint32_t BurstRequestSrc)
Function Description	Stop the DMA burst reading.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle • BurstRequestSrc: TIM DMA Request sources to disable.
Return values	<ul style="list-style-type: none"> • HAL status

41.3.80 HAL_TIM_GenerateEvent

Function Name	HAL_StatusTypeDef HAL_TIM_GenerateEvent (TIM_HandleTypeDef * htim, uint32_t EventSource)
Function Description	Generate a software event.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle • EventSource: specifies the event source. This parameter can be one of the following values: TIM_EVENTSOURCE_UPDATE: Timer update Event source TIM_EVENTSOURCE_CC1: Timer Capture Compare 1 Event source TIM_EVENTSOURCE_CC2: Timer Capture Compare 2 Event source TIM_EVENTSOURCE_CC3: Timer Capture Compare 3 Event source TIM_EVENTSOURCE_CC4: Timer Capture Compare 4 Event source TIM_EVENTSOURCE_TRIGGER: Timer Trigger Event source
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • TIM6 and TIM7 can only generate an update event.

41.3.81 HAL_TIM_ConfigOCrefClear

Function Name	HAL_StatusTypeDef HAL_TIM_ConfigOCrefClear (TIM_HandleTypeDef * htim, TIM_ClearInputConfigTypeDef * sClearInputConfig, uint32_t Channel)
Function Description	Configures the OCRef clear feature.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle • sClearInputConfig: pointer to a TIM_ClearInputConfigTypeDef structure that contains the OCREf clear feature and parameters for the TIM peripheral. • Channel: specifies the TIM Channel This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 TIM_CHANNEL_2: TIM Channel 2 TIM_CHANNEL_3: TIM Channel 3 TIM_CHANNEL_4: TIM Channel 4
Return values	<ul style="list-style-type: none"> • HAL status

41.3.82 HAL_TIM_ConfigClockSource

Function Name	HAL_StatusTypeDef HAL_TIM_ConfigClockSource (TIM_HandleTypeDef * htim, TIM_ClockConfigTypeDef * sClockSourceConfig)
---------------	---

Function Description	Configures the clock source to be used.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle • sClockSourceConfig: pointer to a TIM_ClockConfigTypeDef structure that contains the clock source information for the TIM peripheral.
Return values	<ul style="list-style-type: none"> • HAL status

41.3.83 HAL_TIM_ConfigTI1Input

Function Name	HAL_StatusTypeDef HAL_TIM_ConfigTI1Input (TIM_HandleTypeDef * htim, uint32_t TI1_Selection)
Function Description	Selects the signal connected to the TI1 input: direct from CH1_input or a XOR combination between CH1_input, CH2_input & CH3_input.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle. • TI1_Selection: Indicate whether or not channel 1 is connected to the output of a XOR gate. This parameter can be one of the following values: TIM_TI1SELECTION_CH1: The TIMx_CH1 pin is connected to TI1 inputTIM_TI1SELECTION_XORCOMBINATION: The TIMx_CH1, CH2 and CH3 pins are connected to the TI1 input (XOR combination)
Return values	<ul style="list-style-type: none"> • HAL status

41.3.84 HAL_TIM_SlaveConfigSynchronization

Function Name	HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)
Function Description	Configures the TIM in Slave mode.
Parameters	<ul style="list-style-type: none"> • htim: : TIM handle. • sSlaveConfig: pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the) and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).
Return values	<ul style="list-style-type: none"> • HAL status

41.3.85 HAL_TIM_SlaveConfigSynchronization_IT

Function Name	HAL_StatusTypeDef HAL_TIM_SlaveConfigSynchronization_IT (TIM_HandleTypeDef * htim, TIM_SlaveConfigTypeDef * sSlaveConfig)
Function Description	Configures the TIM in Slave mode in interrupt mode.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle. • sSlaveConfig: pointer to a TIM_SlaveConfigTypeDef structure that contains the selected trigger (internal trigger input, filtered timer input or external trigger input) and the)

and the Slave mode (Disable, Reset, Gated, Trigger, External clock mode 1).

Return values

- HAL status

41.3.86 HAL_TIM_ReadCapturedValue

Function Name **uint32_t HAL_TIM_ReadCapturedValue (TIM_HandleTypeDef * htim, uint32_t Channel)**

Function Description Read the captured value from Capture Compare unit.

Parameters

- **htim**: TIM handle.
- **Channel**: : TIM Channels to be enabled This parameter can be one of the following values: TIM_CHANNEL_1: TIM Channel 1 selectedTIM_CHANNEL_2: TIM Channel 2 selectedTIM_CHANNEL_3: TIM Channel 3 selectedTIM_CHANNEL_4: TIM Channel 4 selected

Return values

- Captured value

41.3.87 HAL_TIM_PeriodElapsedCallback

Function Name **void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)**

Function Description Period elapsed callback in non blocking mode.

Parameters

- **htim**: : TIM handle

Return values

- None

41.3.88 HAL_TIM_OC_DelayElapsedCallback

Function Name **void HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)**

Function Description Output Compare callback in non blocking mode.

Parameters

- **htim**: : TIM OC handle

Return values

- None

41.3.89 HAL_TIM_IC_CaptureCallback

Function Name **void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)**

Function Description Input Capture callback in non blocking mode.

Parameters

- **htim**: : TIM IC handle

Return values

- None

41.3.90 HAL_TIM_PWM_PulseFinishedCallback

Function Name **void HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)**

Function Description PWM Pulse finished callback in non blocking mode.

Parameters	<ul style="list-style-type: none"> • htim : TIM handle
Return values	<ul style="list-style-type: none"> • None

41.3.91 HAL_TIM_TriggerCallback

Function Name	void HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)
Function Description	Hall Trigger detection callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim : TIM handle
Return values	<ul style="list-style-type: none"> • None

41.3.92 HAL_TIM_ErrorCallback

Function Name	void HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)
Function Description	Timer error callback in non blocking mode.
Parameters	<ul style="list-style-type: none"> • htim : TIM handle
Return values	<ul style="list-style-type: none"> • None

41.3.93 HAL_TIM_Base_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Base state.
Parameters	<ul style="list-style-type: none"> • htim: TIM Base handle
Return values	<ul style="list-style-type: none"> • HAL state

41.3.94 HAL_TIM_OC_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM OC state.
Parameters	<ul style="list-style-type: none"> • htim: TIM Output Compare handle
Return values	<ul style="list-style-type: none"> • HAL state

41.3.95 HAL_TIM_PWM_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM PWM state.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle
Return values	<ul style="list-style-type: none"> • HAL state

41.3.96 HAL_TIM_IC_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)
---------------	--

Function Description	Return the TIM Input Capture state.
Parameters	<ul style="list-style-type: none"> • htim: TIM IC handle
Return values	<ul style="list-style-type: none"> • HAL state

41.3.97 HAL_TIM_OnePulse_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM One Pulse Mode state.
Parameters	<ul style="list-style-type: none"> • htim: TIM OPM handle
Return values	<ul style="list-style-type: none"> • HAL state

41.3.98 HAL_TIM_Encoder_GetState

Function Name	HAL_TIM_StateTypeDef HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)
Function Description	Return the TIM Encoder Mode state.
Parameters	<ul style="list-style-type: none"> • htim: TIM Encoder handle
Return values	<ul style="list-style-type: none"> • HAL state

41.3.99 TIM_DMAError

Function Name	void TIM_DMAError (DMA_HandleTypeDef * hdma)
Function Description	TIM DMA error callback.
Parameters	<ul style="list-style-type: none"> • hdma: : pointer to DMA handle.
Return values	<ul style="list-style-type: none"> • None

41.3.100 TIM_DMADelayPulseCplt

Function Name	void TIM_DMADelayPulseCplt (DMA_HandleTypeDef * hdma)
Function Description	TIM DMA Delay Pulse complete callback.
Parameters	<ul style="list-style-type: none"> • hdma: : pointer to DMA handle.
Return values	<ul style="list-style-type: none"> • None

41.3.101 TIM_DMACaptureCplt

Function Name	void TIM_DMACaptureCplt (DMA_HandleTypeDef * hdma)
Function Description	TIM DMA Capture complete callback.
Parameters	<ul style="list-style-type: none"> • hdma: : pointer to DMA handle.
Return values	<ul style="list-style-type: none"> • None

41.4 TIM Firmware driver defines

41.4.1 TIM

TIM Automatic Output Enable

TIM_AUTOMATICOUTPUT_ENABLE

TIM_AUTOMATICOUTPUT_DISABLE

TIM Channel

TIM_CHANNEL_1

TIM_CHANNEL_2

TIM_CHANNEL_3

TIM_CHANNEL_4

TIM_CHANNEL_ALL

TIM Capture/Compare Channel State

TIM_CCx_ENABLE

TIM_CCx_DISABLE

TIM ClearInput Polarity

TIM_CLEARINPUTPOLARITY_INVERTED Polarity for ETRx pin

TIM_CLEARINPUTPOLARITY_NONINVERTED Polarity for ETRx pin

TIM ClearInput Prescaler

TIM_CLEARINPUTPRESCALER_DIV1 No prescaler is used

TIM_CLEARINPUTPRESCALER_DIV2 Prescaler for External ETR pin: Capture performed once every 2 events.

TIM_CLEARINPUTPRESCALER_DIV4 Prescaler for External ETR pin: Capture performed once every 4 events.

TIM_CLEARINPUTPRESCALER_DIV8 Prescaler for External ETR pin: Capture performed once every 8 events.

TIM ClearInput Source

TIM_CLEARINPUTSOURCE_ETR

TIM_CLEARINPUTSOURCE_OCREFCLR

TIM_CLEARINPUTSOURCE_NONE

TIM ClockDivision

TIM_CLOCKDIVISION_DIV1

TIM_CLOCKDIVISION_DIV2

TIM_CLOCKDIVISION_DIV4

TIM Clock Polarity

TIM_CLOCKPOLARITY_INVERTED Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_NONINVERTED Polarity for ETRx clock sources

TIM_CLOCKPOLARITY_RISING Polarity for Tlx clock sources

TIM_CLOCKPOLARITY_FALLING Polarity for Tlx clock sources

TIM_CLOCKPOLARITY_BOTHEDGE Polarity for Tlx clock sources

TIM Clock Prescaler

TIM_CLOCKPRESCALER_DIV1 No prescaler is used

TIM_CLOCKPRESCALER_DIV2 Prescaler for External ETR Clock: Capture performed once every 2 events.

TIM_CLOCKPRESCALER_DIV4 Prescaler for External ETR Clock: Capture performed once every 4 events.

TIM_CLOCKPRESCALER_DIV8 Prescaler for External ETR Clock: Capture performed once every 8 events.

TIM Clock Source

TIM_CLOCKSOURCE_ETRMODE2

TIM_CLOCKSOURCE_INTERNAL

TIM_CLOCKSOURCE_ITR0

TIM_CLOCKSOURCE_ITR1

TIM_CLOCKSOURCE_ITR2

TIM_CLOCKSOURCE_ITR3

TIM_CLOCKSOURCE_TI1ED

TIM_CLOCKSOURCE_TI1

TIM_CLOCKSOURCE_TI2

TIM_CLOCKSOURCE_ETRMODE1

TIM Counter Mode

TIM_COUNTERMODE_UP

TIM_COUNTERMODE_DOWN

TIM_COUNTERMODE_CENTERALIGNED1

TIM_COUNTERMODE_CENTERALIGNED2

TIM_COUNTERMODE_CENTERALIGNED3

TIM DMA Base Address

TIM_DMABASE_CR1

TIM_DMABASE_CR2

TIM_DMABASE_SMCR

TIM_DMABASE_DIER

TIM_DMABASE_SR

TIM_DMABASE_EGR

TIM_DMABASE_CCMR1

TIM_DMABASE_CCMR2

TIM_DMABASE_CCER

TIM_DMABASE_CNT
 TIM_DMABASE_PSC
 TIM_DMABASE_ARR
 TIM_DMABASE_CCR1
 TIM_DMABASE_CCR2
 TIM_DMABASE_CCR3
 TIM_DMABASE_CCR4
 TIM_DMABASE_DCR
 TIM_DMABASE_OR

TIM DMA Burst Length

TIM_DMABURSTLENGTH_1TRANSFER
 TIM_DMABURSTLENGTH_2TRANSFERS
 TIM_DMABURSTLENGTH_3TRANSFERS
 TIM_DMABURSTLENGTH_4TRANSFERS
 TIM_DMABURSTLENGTH_5TRANSFERS
 TIM_DMABURSTLENGTH_6TRANSFERS
 TIM_DMABURSTLENGTH_7TRANSFERS
 TIM_DMABURSTLENGTH_8TRANSFERS
 TIM_DMABURSTLENGTH_9TRANSFERS
 TIM_DMABURSTLENGTH_10TRANSFERS
 TIM_DMABURSTLENGTH_11TRANSFERS
 TIM_DMABURSTLENGTH_12TRANSFERS
 TIM_DMABURSTLENGTH_13TRANSFERS
 TIM_DMABURSTLENGTH_14TRANSFERS
 TIM_DMABURSTLENGTH_15TRANSFERS
 TIM_DMABURSTLENGTH_16TRANSFERS
 TIM_DMABURSTLENGTH_17TRANSFERS
 TIM_DMABURSTLENGTH_18TRANSFERS

TIM DMA Handle Index

TIM_DMA_ID_UPDATE	Index of the DMA handle used for Update DMA requests
TIM_DMA_ID_CC1	Index of the DMA handle used for Capture/Compare 1 DMA requests
TIM_DMA_ID_CC2	Index of the DMA handle used for Capture/Compare 2 DMA requests
TIM_DMA_ID_CC3	Index of the DMA handle used for Capture/Compare 3 DMA requests
TIM_DMA_ID_CC4	Index of the DMA handle used for Capture/Compare 4 DMA requests

TIM_DMA_ID_TRIGGER Index of the DMA handle used for Trigger DMA requests

TIM DMA Sources

TIM_DMA_UPDATE

TIM_DMA_CC1

TIM_DMA_CC2

TIM_DMA_CC3

TIM_DMA_CC4

TIM_DMA_TRIGGER

TIM Encoder Mode

TIM_ENCODERMODE_TI1

TIM_ENCODERMODE_TI2

TIM_ENCODERMODE_TI12

TIM ETR Polarity

TIM_ETRPOLARITY_INVERTED Polarity for ETR source

TIM_ETRPOLARITY_NONINVERTED Polarity for ETR source

TIM ETR Prescaler

TIM_ETRPRESCALER_DIV1 No prescaler is used

TIM_ETRPRESCALER_DIV2 ETR input source is divided by 2

TIM_ETRPRESCALER_DIV4 ETR input source is divided by 4

TIM_ETRPRESCALER_DIV8 ETR input source is divided by 8

TIM Event Source

TIM_EVENTSOURCE_UPDATE

TIM_EVENTSOURCE_CC1

TIM_EVENTSOURCE_CC2

TIM_EVENTSOURCE_CC3

TIM_EVENTSOURCE_CC4

TIM_EVENTSOURCE_TRIGGER

TIM Exported Macros

__HAL_TIM_RESET_HANDLE_STATE

Description:

- Reset TIM handle state.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

__HAL_TIM_ENABLE

Description:

- Enable the TIM peripheral.

`__HAL_TIM_DISABLE`

Parameters:

- `__HANDLE__`: TIM handle

Return value:

- None

Description:

- Disable the TIM peripheral.

Parameters:

- `__HANDLE__`: TIM handle

Return value:

- None

Description:

- Enables the specified TIM interrupt.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__INTERRUPT__`: specifies the TIM interrupt source to enable. This parameter can be one of the following values:
 - `TIM_IT_UPDATE`: Update interrupt
 - `TIM_IT_CC1`: Capture/Compare 1 interrupt
 - `TIM_IT_CC2`: Capture/Compare 2 interrupt
 - `TIM_IT_CC3`: Capture/Compare 3 interrupt
 - `TIM_IT_CC4`: Capture/Compare 4 interrupt
 - `TIM_IT_COM`: Commutation interrupt
 - `TIM_IT_TRIGGER`: Trigger interrupt

Return value:

- None

Description:

- Disables the specified TIM interrupt.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__INTERRUPT__`: specifies the TIM interrupt source to disable. This parameter can be one of the following values:
 - `TIM_IT_UPDATE`: Update interrupt
 - `TIM_IT_CC1`: Capture/Compare 1 interrupt
 - `TIM_IT_CC2`: Capture/Compare 2 interrupt
 - `TIM_IT_CC3`: Capture/Compare 3 interrupt
 - `TIM_IT_CC4`: Capture/Compare 4 interrupt
 - `TIM_IT_COM`: Commutation interrupt
 - `TIM_IT_TRIGGER`: Trigger interrupt

Return value:

- None

`__HAL_TIM_ENABLE_IT`

`__HAL_TIM_DISABLE_IT`

__HAL_TIM_ENABLE_DMA**Description:**

- Enables the specified DMA request.

Parameters:

- **__HANDLE__**: specifies the TIM Handle.
- **__DMA__**: specifies the TIM DMA request to enable. This parameter can be one of the following values:
 - TIM_DMA_UPDATE: Update DMA request
 - TIM_DMA_CC1: Capture/Compare 1 DMA request
 - TIM_DMA_CC2: Capture/Compare 2 DMA request
 - TIM_DMA_CC3: Capture/Compare 3 DMA request
 - TIM_DMA_CC4: Capture/Compare 4 DMA request
 - TIM_DMA_COM: Commutation DMA request
 - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_DISABLE_DMA**Description:**

- Disables the specified DMA request.

Parameters:

- **__HANDLE__**: specifies the TIM Handle.
- **__DMA__**: specifies the TIM DMA request to disable. This parameter can be one of the following values:
 - TIM_DMA_UPDATE: Update DMA request
 - TIM_DMA_CC1: Capture/Compare 1 DMA request
 - TIM_DMA_CC2: Capture/Compare 2 DMA request
 - TIM_DMA_CC3: Capture/Compare 3 DMA request
 - TIM_DMA_CC4: Capture/Compare 4 DMA request
 - TIM_DMA_COM: Commutation DMA request
 - TIM_DMA_TRIGGER: Trigger DMA request

Return value:

- None

__HAL_TIM_GET_FLAG**Description:**

- Checks whether the specified TIM interrupt flag is set or not.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__FLAG__`: specifies the TIM interrupt flag to check. This parameter can be one of the following values:
 - `TIM_FLAG_UPDATE`: Update interrupt flag
 - `TIM_FLAG_CC1`: Capture/Compare 1 interrupt flag
 - `TIM_FLAG_CC2`: Capture/Compare 2 interrupt flag
 - `TIM_FLAG_CC3`: Capture/Compare 3 interrupt flag
 - `TIM_FLAG_CC4`: Capture/Compare 4 interrupt flag
 - `TIM_FLAG_COM`: Commutation interrupt flag
 - `TIM_FLAG_TRIGGER`: Trigger interrupt flag
 - `TIM_FLAG_CC1OF`: Capture/Compare 1 overcapture flag
 - `TIM_FLAG_CC2OF`: Capture/Compare 2 overcapture flag
 - `TIM_FLAG_CC3OF`: Capture/Compare 3 overcapture flag
 - `TIM_FLAG_CC4OF`: Capture/Compare 4 overcapture flag

Return value:

- The: new state of `__FLAG__` (TRUE or FALSE).

`__HAL_TIM_CLEAR_FLAG`**Description:**

- Clears the specified TIM interrupt flag.

Parameters:

- `__HANDLE__`: specifies the TIM Handle.
- `__FLAG__`: specifies the TIM interrupt flag to clear. This parameter can be one of the following values:
 - `TIM_FLAG_UPDATE`: Update interrupt flag
 - `TIM_FLAG_CC1`: Capture/Compare 1 interrupt flag
 - `TIM_FLAG_CC2`: Capture/Compare 2 interrupt flag
 - `TIM_FLAG_CC3`: Capture/Compare 3 interrupt flag
 - `TIM_FLAG_CC4`: Capture/Compare 4 interrupt flag
 - `TIM_FLAG_COM`: Commutation interrupt flag
 - `TIM_FLAG_TRIGGER`: Trigger interrupt flag
 - `TIM_FLAG_CC1OF`: Capture/Compare 1 overcapture flag

- TIM_FLAG_CC2OF: Capture/Compare 2 overcapture flag
- TIM_FLAG_CC3OF: Capture/Compare 3 overcapture flag
- TIM_FLAG_CC4OF: Capture/Compare 4 overcapture flag

Return value:

- The: new state of __FLAG__ (TRUE or FALSE).

__HAL_TIM_GET_IT_SOURCE**Description:**

- Checks whether the specified TIM interrupt has occurred or not.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the TIM interrupt source to check.

Return value:

- The: state of TIM_IT (SET or RESET).

__HAL_TIM_CLEAR_IT**Description:**

- Clear the TIM interrupt pending bits.

Parameters:

- __HANDLE__: TIM handle
- __INTERRUPT__: specifies the interrupt pending bit to clear.

Return value:

- None

__HAL_TIM_IS_TIM_COUNTING_DOWN**Description:**

- Indicates whether or not the TIM Counter is used as downcounter.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- False: (Counter used as upcounter) or True (Counter used as downcounter)

Notes:

- This macro is particularly usefull to get the counting mode when the timer operates in Center-aligned mode or Encoder mode.

__HAL_TIM_SET_PRESCALER**Description:**

- Sets the TIM active prescaler register value on update event.

Parameters:

__HAL_TIM_SET_COMPARE

- **__HANDLE__**: TIM handle.
- **__PRESC__**: specifies the active prescaler register new value.

Return value:

- None

Description:

- Sets the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.

Parameters:

- **__HANDLE__**: TIM handle.
- **__CHANNEL__**: : TIM Channels to be configured. This parameter can be one of the following values:
 - **TIM_CHANNEL_1**: TIM Channel 1 selected
 - **TIM_CHANNEL_2**: TIM Channel 2 selected
 - **TIM_CHANNEL_3**: TIM Channel 3 selected
 - **TIM_CHANNEL_4**: TIM Channel 4 selected
- **__COMPARE__**: specifies the Capture Compare register new value.

Return value:

- None

__HAL_TIM_GET_COMPARE**Description:**

- Gets the TIM Capture Compare Register value on runtime.

Parameters:

- **__HANDLE__**: TIM handle.
- **__CHANNEL__**: : TIM Channel associated with the capture compare register This parameter can be one of the following values:
 - **TIM_CHANNEL_1**: get capture/compare 1 register value
 - **TIM_CHANNEL_2**: get capture/compare 2 register value
 - **TIM_CHANNEL_3**: get capture/compare 3 register value
 - **TIM_CHANNEL_4**: get capture/compare 4 register value

Return value:

- None

__HAL_TIM_SET_COUNTER**Description:**

- Sets the TIM Counter Register value on

runtime.

Parameters:

- `__HANDLE__`: TIM handle.
- `__COUNTER__`: specifies the Counter register new value.

Return value:

- None

`__HAL_TIM_GET_COUNTER`**Description:**

- Gets the TIM Counter Register value on runtime.

Parameters:

- `__HANDLE__`: TIM handle.

Return value:

- None

`__HAL_TIM_SET_AUTORELOAD`**Description:**

- Sets the TIM Autoreload Register value on runtime without calling another time any Init function.

Parameters:

- `__HANDLE__`: TIM handle.
- `__AUTORELOAD__`: specifies the Counter register new value.

Return value:

- None

`__HAL_TIM_GET_AUTORELOAD`**Description:**

- Gets the TIM Autoreload Register value on runtime.

Parameters:

- `__HANDLE__`: TIM handle.

Return value:

- None

`__HAL_TIM_SET_CLOCKDIVISION`**Description:**

- Sets the TIM Clock Division value on runtime without calling another time any Init function.

Parameters:

- `__HANDLE__`: TIM handle.
- `__CKD__`: specifies the clock division value. This parameter can be one of the following value:
 - `TIM_CLOCKDIVISION_DIV1`
 - `TIM_CLOCKDIVISION_DIV2`

– TIM_CLOCKDIVISION_DIV4

Return value:

- None

`__HAL_TIM_GET_CLOCKDIVISION`

Description:

- Gets the TIM Clock Division value on runtime.

Parameters:

- `__HANDLE__`: TIM handle.

Return value:

- None

`__HAL_TIM_SET_ICPRESCALER`

Description:

- Sets the TIM Input Capture prescaler on runtime without calling another time

Parameters:

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:
 - TIM_CHANNEL_1: TIM Channel 1 selected
 - TIM_CHANNEL_2: TIM Channel 2 selected
 - TIM_CHANNEL_3: TIM Channel 3 selected
 - TIM_CHANNEL_4: TIM Channel 4 selected
- `__ICPSC__`: specifies the Input Capture4 prescaler new value. This parameter can be one of the following values:
 - TIM_ICPSC_DIV1: no prescaler
 - TIM_ICPSC_DIV2: capture is done once every 2 events
 - TIM_ICPSC_DIV4: capture is done once every 4 events
 - TIM_ICPSC_DIV8: capture is done once every 8 events

Return value:

- None

`__HAL_TIM_GET_ICPRESCALER`

Description:

- Gets the TIM Input Capture prescaler on runtime.

Parameters:

- `__HANDLE__`: TIM handle.
- `__CHANNEL__`: TIM Channels to be configured. This parameter can be one of the following values:

- TIM_CHANNEL_1: get input capture 1 prescaler value
- TIM_CHANNEL_2: get input capture 2 prescaler value
- TIM_CHANNEL_3: get input capture 3 prescaler value
- TIM_CHANNEL_4: get input capture 4 prescaler value

Return value:

- None

__HAL_TIM_URS_ENABLE**Description:**

- Set the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the USR bit of the TIMx_CR1 register is set, only counter overflow/underflow generates an update interrupt or DMA request (if enabled)

__HAL_TIM_URS_DISABLE**Description:**

- Reset the Update Request Source (URS) bit of the TIMx_CR1 register.

Parameters:

- __HANDLE__: TIM handle.

Return value:

- None

Notes:

- When the USR bit of the TIMx_CR1 register is reset, any of the following events generate an update interrupt or DMA request (if enabled): (+) Counter overflow/underflow (+) Setting the UG bit (+) Update generation through the slave mode controller

__HAL_TIM_SET_CAPTUREPOLARITY**Description:**

- Sets the TIM Capture x input polarity on runtime.

Parameters:

- __HANDLE__: TIM handle.
- __CHANNEL__: TIM Channels to be configured. This parameter can be one of the following values:

- TIM_CHANNEL_1: TIM Channel 1 selected
- TIM_CHANNEL_2: TIM Channel 2 selected
- TIM_CHANNEL_3: TIM Channel 3 selected
- TIM_CHANNEL_4: TIM Channel 4 selected
- __POLARITY__: Polarity for TIx source
 - TIM_INPUTCHANNELPOLARITY_RISING : Rising Edge
 - TIM_INPUTCHANNELPOLARITY_FALLING : Falling Edge
 - TIM_INPUTCHANNELPOLARITY_BOTHEDGE : Rising and Falling Edge

Return value:

- None

Notes:

- The polarity TIM_INPUTCHANNELPOLARITY_BOTHEDGE is not authorized for TIM Channel 4.

TIM Flag Definition

TIM_FLAG_UPDATE

TIM_FLAG_CC1

TIM_FLAG_CC2

TIM_FLAG_CC3

TIM_FLAG_CC4

TIM_FLAG_TRIGGER

TIM_FLAG_CC1OF

TIM_FLAG_CC2OF

TIM_FLAG_CC3OF

TIM_FLAG_CC4OF

TIM Input Capture Polarity

TIM_ICPOLARITY_RISING

TIM_ICPOLARITY_FALLING

TIM_ICPOLARITY_BOTHEDGE

TIM Input Capture Prescaler

TIM_ICPSC_DIV1 Capture performed each time an edge is detected on the capture input

TIM_ICPSC_DIV2 Capture performed once every 2 events

TIM_ICPSC_DIV4 Capture performed once every 4 events

TIM_ICPSC_DIV8 Capture performed once every 8 events

TIM Input Capture Selection

TIM_ICSELECTION_DIRECTTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively
TIM_ICSELECTION_INDIRECTTI	TIM Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively
TIM_ICSELECTION_TRC	TIM Input 1, 2, 3 or 4 is selected to be connected to TRC

TIM Input Channel Polarity

TIM_INPUTCHANNELPOLARITY_RISING	Polarity for Tlx source
TIM_INPUTCHANNELPOLARITY_FALLING	Polarity for Tlx source
TIM_INPUTCHANNELPOLARITY_BOTHEDGE	Polarity for Tlx source

TIM Interrupt Definition

TIM_IT_UPDATE
TIM_IT_CC1
TIM_IT_CC2
TIM_IT_CC3
TIM_IT_CC4
TIM_IT_TRIGGER

TIM Lock level

TIM_LOCKLEVEL_OFF
TIM_LOCKLEVEL_1
TIM_LOCKLEVEL_2
TIM_LOCKLEVEL_3

TIM Master Mode Selection

TIM_TRGO_RESET
TIM_TRGO_ENABLE
TIM_TRGO_UPDATE
TIM_TRGO_OC1
TIM_TRGO_OC1REF
TIM_TRGO_OC2REF
TIM_TRGO_OC3REF
TIM_TRGO_OC4REF

TIM Master Slave Mode

TIM_MASTERSLAVEMODE_ENABLE
TIM_MASTERSLAVEMODE_DISABLE

TIM One Pulse Mode

TIM_OPMODE_SINGLE

TIM_OPMODE_REPETITIVE

TIM OSSI Off State Selection for Idle mode state

TIM_OSSI_ENABLE

TIM_OSSI_DISABLE

TIM OSSR Off State Selection for Run mode state

TIM_OSSR_ENABLE

TIM_OSSR_DISABLE

TIM Output Compare and PWM modes

TIM_OCMODE_TIMING

TIM_OCMODE_ACTIVE

TIM_OCMODE_INACTIVE

TIM_OCMODE_TOGGLE

TIM_OCMODE_PWM1

TIM_OCMODE_PWM2

TIM_OCMODE_FORCED_ACTIVE

TIM_OCMODE_FORCED_INACTIVE

TIM Output Compare Idle State

TIM_OCIDLESTATE_SET

TIM_OCIDLESTATE_RESET

TIM Output Compare Polarity

TIM_OCPOLARITY_HIGH

TIM_OCPOLARITY_LOW

TIM Output Fast State

TIM_OCFAST_DISABLE

TIM_OCFAST_ENABLE

TIM Private Constants

TIM_CCER_CCxE_MASK

TIM Private Macros

IS_TIM_COUNTER_MODE

IS_TIM_CLOCKDIVISION_DIV

IS_TIM_PWM_MODE

IS_TIM_OC_MODE

IS_TIM_FAST_STATE

IS_TIM_OC_POLARITY

IS_TIM_OCIDLE_STATE

IS_TIM_CHANNELS

IS_TIM_OPM_CHANNELS
IS_TIM_IC_POLARITY
IS_TIM_IC_SELECTION
IS_TIM_IC_PRESCALER
IS_TIM_OPM_MODE
IS_TIM_ENCODER_MODE
IS_TIM_DMA_SOURCE
IS_TIM_EVENT_SOURCE
IS_TIM_CLOCKSOURCE
IS_TIM_CLOCKPOLARITY
IS_TIM_CLOCKPRESCALER
IS_TIM_CLOCKFILTER
IS_TIM_CLEARINPUT_SOURCE
IS_TIM_CLEARINPUT_POLARITY
IS_TIM_CLEARINPUT_PRESCALER
IS_TIM_CLEARINPUT_FILTER
IS_TIM_OSSR_STATE
IS_TIM_OSSI_STATE
IS_TIM_LOCK_LEVEL
IS_TIM_AUTOMATIC_OUTPUT_STATE
IS_TIM_TRGO_SOURCE
IS_TIM_SLAVE_MODE
IS_TIM_MSM_STATE
IS_TIM_TRIGGER_SELECTION
IS_TIM_INTERNAL_TRIGGEREVENT_SELECTION
IS_TIM_TRIGGERPOLARITY
IS_TIM_TRIGGERPRESCALER
IS_TIM_TRIGGERFILTER
IS_TIM_TI1SELECTION
IS_TIM_DMA_BASE
IS_TIM_DMA_LENGTH
IS_TIM_IC_FILTER
TIM_SET_ICPRESCALERVALUE

Description:

- Set TIM IC prescaler.

Parameters:

- `__HANDLE__`: TIM handle
- `__CHANNEL__`: specifies TIM

TIM_RESET_ICPRESCALERVALUE

Channel

- `__ICPSC__`: specifies the prescaler value.

Return value:

- None

Description:

- Reset TIM IC prescaler.

Parameters:

- `__HANDLE__`: TIM handle
- `__CHANNEL__`: specifies TIM Channel

Return value:

- None

Description:

- Set TIM IC polarity.

Parameters:

- `__HANDLE__`: TIM handle
- `__CHANNEL__`: specifies TIM Channel
- `__POLARITY__`: specifies TIM Channel Polarity

Return value:

- None

Description:

- Reset TIM IC polarity.

Parameters:

- `__HANDLE__`: TIM handle
- `__CHANNEL__`: specifies TIM Channel

Return value:

- None

TIM_SET_CAPTUREPOLARITY

TIM_RESET_CAPTUREPOLARITY

TIM Select

TIM_SELECT_NONE None selected
 TIM_SELECT_TIM2 Timer 2 selected
 TIM_SELECT_TIM3 Timer 3 selected
 TIM_SELECT_TIM4 Timer 4 selected

IS_RI_TIM

TIM Slave Mode

TIM_SLAVEMODE_DISABLE

TIM_SLAVEMODE_RESET

TIM_SLAVEMODE_GATED

TIM_SLAVEMODE_TRIGGER

TIM_SLAVEMODE_EXTERNAL1

TIM TI1 Input Selection

TIM_TI1SELECTION_CH1

TIM_TI1SELECTION_XORCOMBINATION

TIM Trigger Polarity

TIM_TRIGGERPOLARITY_INVERTED Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_NONINVERTED Polarity for ETRx trigger sources

TIM_TRIGGERPOLARITY_RISING Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_FALLING Polarity for TIxFPx or TI1_ED trigger sources

TIM_TRIGGERPOLARITY_BOTHEDGE Polarity for TIxFPx or TI1_ED trigger sources

TIM Trigger Prescaler

TIM_TRIGGERPRESCALER_DIV1 No prescaler is used

TIM_TRIGGERPRESCALER_DIV2 Prescaler for External ETR Trigger: Capture performed once every 2 events.

TIM_TRIGGERPRESCALER_DIV4 Prescaler for External ETR Trigger: Capture performed once every 4 events.

TIM_TRIGGERPRESCALER_DIV8 Prescaler for External ETR Trigger: Capture performed once every 8 events.

TIM Trigger Selection

TIM_TS_ITR0

TIM_TS_ITR1

TIM_TS_ITR2

TIM_TS_ITR3

TIM_TS_TI1F_ED

TIM_TS_TI1FP1

TIM_TS_TI2FP2

TIM_TS_ETRF

TIM_TS_NONE

42 HAL TIM Extension Driver

42.1 HAL TIM Extension Driver

42.2 TIMEx Firmware driver registers structures

42.2.1 TIM_MasterConfigTypeDef

Data Fields

- *uint32_t MasterOutputTrigger*
- *uint32_t MasterSlaveMode*

Field Documentation

- *uint32_t TIM_MasterConfigTypeDef::MasterOutputTrigger*
Trigger output (TRGO) selection This parameter can be a value of [TIM_Master_Mode_Selection](#)
- *uint32_t TIM_MasterConfigTypeDef::MasterSlaveMode*
Master/slave mode selection This parameter can be a value of [TIM_Master_Slave_Mode](#)

42.3 TIMEx Firmware driver API description

42.3.1 TIMER Extended features

The Timer Extension features include:

1. Synchronization circuit to control the timer with external signals and to interconnect several timers together.
2. Timer remapping capabilities configuration

42.3.2 Peripheral Control functions

This section provides functions allowing to:

- Configure Master synchronization.
- Configure timer remapping capabilities.

This section contains the following APIs:

- [HAL_TIMEx_MasterConfigSynchronization\(\)](#)
- [HAL_TIMEx_RemapConfig\(\)](#)

42.3.3 HAL_TIMEx_MasterConfigSynchronization

Function Name	HAL_StatusTypeDef
	HAL_TIMEx_MasterConfigSynchronization
	(TIM_HandleTypeDef * htim, TIM_MasterConfigTypeDef * sMasterConfig)

Function Description	Configures the TIM in master mode.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle. • sMasterConfig: pointer to a TIM_MasterConfigTypeDef structure that contains the selected trigger output (TRGO) and the Master/Slave mode.
Return values	<ul style="list-style-type: none"> • HAL status

42.3.4 HAL_TIMEx_RemapConfig

Function Name	HAL_StatusTypeDef HAL_TIMEx_RemapConfig (TIM_HandleTypeDef * htim, uint32_t Remap)
Function Description	Configures the TIM2/TIM3/TIM9/TIM10/TIM11 Remapping input capabilities.
Parameters	<ul style="list-style-type: none"> • htim: TIM handle. • Remap: specifies the TIM remapping source. This parameter is a combination of the following values depending on TIM instance.
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • For TIM2, the parameter can have the following values: TIM_TIM2_ITR1_TIM10_OC: TIM2 ITR1 input is connected to TIM10 OC TIM_TIM2_ITR1_TIM5_TGO: TIM2 ITR1 input is connected to TIM5 TGO • For TIM3, the parameter can have the following values: TIM_TIM3_ITR2_TIM11_OC: TIM3 ITR2 input is connected to TIM11 OC TIM_TIM3_ITR2_TIM5_TGO: TIM3 ITR2 input is connected to TIM5 TGO • For TIM9, the parameter is a combination of 2 fields (field1 field2): • For TIM9, the field1 can have the following values: TIM_TIM9_ITR1_TIM3_TGO: TIM9 ITR1 input is connected to TIM3 TGO TIM_TIM9_ITR1_TS: TIM9 ITR1 input is connected to touch sensing I/O • For TIM9, the field2 can have the following values: TIM_TIM9_GPIO: TIM9 Channel1 is connected to GPIO TIM_TIM9_LSE: TIM9 Channel1 is connected to LSE internal clock TIM_TIM9_GPIO1: TIM9 Channel1 is connected to GPIO TIM_TIM9_GPIO2: TIM9 Channel1 is connected to GPIO • For TIM10, the parameter is a combination of 3 fields (field1 field2 field3): • For TIM10, the field1 can have the following values: TIM_TIM10_TI1RMP: TIM10 Channel 1 depends on TI1_RMP TIM_TIM10_RI: TIM10 Channel 1 is connected to RI • For TIM10, the field2 can have the following values: TIM_TIM10_ETR_LSE: TIM10 ETR input is connected to LSE clock TIM_TIM10_ETR_TIM9_TGO: TIM10 ETR input is connected to TIM9 TGO • For TIM10, the field3 can have the following values: TIM_TIM10_GPIO: TIM10 Channel1 is connected to GPIO TIM_TIM10_LSI: TIM10 Channel1 is connected to LSI internal

clock TIM_TIM10_LSE: TIM10 Channel1 is connected to LSE
 internal clock TIM_TIM10_RTC: TIM10 Channel1 is connected to RTC wakeup interrupt

- For TIM11, the parameter is a combination of 3 fields (field1 | field2 | field3):
- For TIM11, the field1 can have the following values:
 TIM_TIM11_TI1RMP: TIM11 Channel 1 depends on TI1_RMP
 TIM_TIM11_RI: TIM11 Channel 1 is connected to RI
- For TIM11, the field2 can have the following values:
 TIM_TIM11_ETR_LSE: TIM11 ETR input is connected to LSE
 clock TIM_TIM11_ETR_TIM9_TGO: TIM11 ETR input is connected to TIM9 TGO
- For TIM11, the field3 can have the following values:
 TIM_TIM11_GPIO: TIM11 Channel1 is connected to GPIO
 TIM_TIM11_MSI: TIM11 Channel1 is connected to MSI
 internal clock TIM_TIM11_HSE_RTC: TIM11 Channel1 is connected to HSE_RTC
 clock TIM_TIM11_GPIO1: TIM11 Channel1 is connected to GPIO

42.4 TIMEx Firmware driver defines

42.4.1 TIMEx

TIMEx Remap

TIM_TIM2_ITR1_TIM10_OC	TIM2 ITR1 input is connected to TIM10 OC
TIM_TIM2_ITR1_TIM5_TGO	TIM2 ITR1 input is connected to TIM5 TGO
TIM_TIM3_ITR2_TIM11_OC	TIM3 ITR2 input is connected to TIM11 OC
TIM_TIM3_ITR2_TIM5_TGO	TIM3 ITR2 input is connected to TIM5 TGO
TIM_TIM9_ITR1_TIM3_TGO	TIM9 ITR1 input is connected to TIM3 TGO
TIM_TIM9_ITR1_TS	TIM9 ITR1 input is connected to touch sensing I/O
TIM_TIM9_GPIO	TIM9 Channel1 is connected to GPIO
TIM_TIM9_LSE	TIM9 Channel1 is connected to LSE internal clock
TIM_TIM9_GPIO1	TIM9 Channel1 is connected to GPIO
TIM_TIM9_GPIO2	TIM9 Channel1 is connected to GPIO
TIM_TIM10_TI1RMP	TIM10 Channel 1 depends on TI1_RMP
TIM_TIM10_RI	TIM10 Channel 1 is connected to RI
TIM_TIM10_ETR_LSE	TIM10 ETR input is connected to LSE clock
TIM_TIM10_ETR_TIM9_TGO	TIM10 ETR input is connected to TIM9 TGO
TIM_TIM10_GPIO	TIM10 Channel1 is connected to GPIO
TIM_TIM10_LSI	TIM10 Channel1 is connected to LSI internal clock
TIM_TIM10_LSE	TIM10 Channel1 is connected to LSE internal clock
TIM_TIM10_RTC	TIM10 Channel1 is connected to RTC wakeup interrupt
TIM_TIM11_TI1RMP	TIM11 Channel 1 depends on TI1_RMP

TIM_TIM11_RI	TIM11 Channel 1 is connected to RI
TIM_TIM11_ETR_LSE	TIM11 ETR input is connected to LSE clock
TIM_TIM11_ETR_TIM9_TGO	TIM11 ETR input is connected to TIM9 TGO
TIM_TIM11_GPIO	TIM11 Channel1 is connected to GPIO
TIM_TIM11_MSI	TIM11 Channel1 is connected to MSI internal clock
TIM_TIM11_HSE_RTC	TIM11 Channel1 is connected to HSE_RTC clock
TIM_TIM11_GPIO1	TIM11 Channel1 is connected to GPIO
IS_TIM_REMAP	

43 HAL UART Generic Driver

43.1 HAL UART Generic Driver

43.2 UART Firmware driver registers structures

43.2.1 UART_InitTypeDef

Data Fields

- *uint32_t BaudRate*
- *uint32_t WordLength*
- *uint32_t StopBits*
- *uint32_t Parity*
- *uint32_t Mode*
- *uint32_t HwFlowCtl*
- *uint32_t OverSampling*

Field Documentation

- ***uint32_t UART_InitTypeDef::BaudRate***
This member configures the UART communication baud rate. The baud rate is computed using the following formula: $\text{IntegerDivider} = ((\text{PCLKx}) / (8 * (\text{OVR8} + 1) * (\text{huart->Init.BaudRate})))$ $\text{FractionalDivider} = ((\text{IntegerDivider} - ((\text{uint32_t}) \text{IntegerDivider})) * 8 * (\text{OVR8} + 1)) + 0.5$ Where OVR8 is the "oversampling by 8 mode" configuration bit in the CR1 register.
- ***uint32_t UART_InitTypeDef::WordLength***
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [UART_Word_Length](#)
- ***uint32_t UART_InitTypeDef::StopBits***
Specifies the number of stop bits transmitted. This parameter can be a value of [UART_Stop_Bits](#)
- ***uint32_t UART_InitTypeDef::Parity***
Specifies the parity mode. This parameter can be a value of [UART_Parity](#)
Note:When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t UART_InitTypeDef::Mode***
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [UART_Mode](#)
- ***uint32_t UART_InitTypeDef::HwFlowCtl***
Specifies whether the hardware flow control mode is enabled or disabled. This parameter can be a value of [UART_Hardware_Flow_Control](#)
- ***uint32_t UART_InitTypeDef::OverSampling***
Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to fPCLK/8). This parameter can be a value of [UART_Over_Sampling](#)

43.2.2 UART_HandleTypeDef

Data Fields

- **USART_TypeDef * Instance**
- **UART_InitTypeDef Init**
- **uint8_t * pTxBuffPtr**
- **uint16_t TxXferSize**
- **uint16_t TxXferCount**
- **uint8_t * pRxBuffPtr**
- **uint16_t RxXferSize**
- **uint16_t RxXferCount**
- **DMA_HandleTypeDef * hdmatx**
- **DMA_HandleTypeDef * hdmarx**
- **HAL_LockTypeDef Lock**
- **__IO HAL_UART_StateTypeDef State**
- **__IO uint32_t ErrorCode**

Field Documentation

- **USART_TypeDef* UART_HandleTypeDef::Instance**
UART registers base address
- **UART_InitTypeDef UART_HandleTypeDef::Init**
UART communication parameters
- **uint8_t* UART_HandleTypeDef::pTxBuffPtr**
Pointer to UART Tx transfer Buffer
- **uint16_t UART_HandleTypeDef::TxXferSize**
UART Tx Transfer size
- **uint16_t UART_HandleTypeDef::TxXferCount**
UART Tx Transfer Counter
- **uint8_t* UART_HandleTypeDef::pRxBuffPtr**
Pointer to UART Rx transfer Buffer
- **uint16_t UART_HandleTypeDef::RxXferSize**
UART Rx Transfer size
- **uint16_t UART_HandleTypeDef::RxXferCount**
UART Rx Transfer Counter
- **DMA_HandleTypeDef* UART_HandleTypeDef::hdmatx**
UART Tx DMA Handle parameters
- **DMA_HandleTypeDef* UART_HandleTypeDef::hdmarx**
UART Rx DMA Handle parameters
- **HAL_LockTypeDef UART_HandleTypeDef::Lock**
Locking object
- **__IO HAL_UART_StateTypeDef UART_HandleTypeDef::State**
UART communication state
- **__IO uint32_t UART_HandleTypeDef::ErrorCode**
UART Error code

43.3 UART Firmware driver API description

43.3.1 How to use this driver

The UART HAL driver can be used as follows:

1. Declare a UART_HandleTypeDef handle structure.
2. Initialize the UART low level resources by implementing the HAL_UART_MspInit() API:
 - a. Enable the USARTx interface clock.
 - b. UART pins configuration:
 - Enable the clock for the UART GPIOs.
 - Configure the UART pins as alternate function pull-up.
 - c. NVIC configuration if you need to use interrupt process (HAL_UART_Transmit_IT() and HAL_UART_Receive_IT() APIs):
 - Configure the USARTx interrupt priority.
 - Enable the NVIC USART IRQ handle.
 - d. DMA Configuration if you need to use DMA process (HAL_UART_Transmit_DMA() and HAL_UART_Receive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx channel.
 - Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
 - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the UART Init structure.
4. For the UART asynchronous mode, initialize the UART registers by calling the HAL_UART_Init() API.
5. For the UART Half duplex mode, initialize the UART registers by calling the HAL_HalfDuplex_Init() API.
6. For the LIN mode, initialize the UART registers by calling the HAL_LIN_Init() API.
7. For the Multi-Processor mode, initialize the UART registers by calling the HAL_MultiProcessor_Init() API.



The specific UART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive process.



These APIs (HAL_UART_Init() and HAL_HalfDuplex_Init()) configure also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_UART_MspInit() API.

Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_UART_Transmit()
- Receive an amount of data in blocking mode using HAL_UART_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_UART_Transmit_IT()
- At transmission end of transfer HAL_UART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_UART_Receive_IT()
- At reception end of transfer HAL_UART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxCpltCallback
- In case of transfer Error, HAL_UART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_UART_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_UART_Transmit_DMA()
- At transmission end of half transfer HAL_UART_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxHalfCpltCallback
- At transmission end of transfer HAL_UART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_UART_Receive_DMA()
- At reception end of half transfer HAL_UART_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxHalfCpltCallback
- At reception end of transfer HAL_UART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_UART_RxCpltCallback
- In case of transfer Error, HAL_UART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_UART_ErrorCallback
- Pause the DMA Transfer using HAL_UART_DMAPause()
- Resume the DMA Transfer using HAL_UART_DMAResume()
- Stop the DMA Transfer using HAL_UART_DMAStop()

UART HAL driver macros list

Below the list of most used macros in UART HAL driver.

- __HAL_UART_ENABLE: Enable the UART peripheral
- __HAL_UART_DISABLE: Disable the UART peripheral
- __HAL_UART_GET_FLAG : Check whether the specified UART flag is set or not
- __HAL_UART_CLEAR_FLAG : Clear the specified UART pending flag
- __HAL_UART_ENABLE_IT: Enable the specified UART interrupt
- __HAL_UART_DISABLE_IT: Disable the specified UART interrupt
- __HAL_UART_GET_IT_SOURCE: Check whether the specified UART interrupt has occurred or not



You can refer to the UART HAL driver header file for more useful macros

43.3.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USARTx or the UARTy in asynchronous mode.

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Stop Bit
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), the possible UART frame formats are as listed in [Table 19: "OPAMPs inverting/non-inverting inputs for STM32L1 devices"](#)
 - Hardware flow control
 - Receiver/transmitter modes
 - Over Sampling Methode

Table 24: UART frame formats

M bit	PCE bit	UART frame
0	0	SB 8 bit data STB
0	1	SB 7 bit data PB STB
1	0	SB 9 bit data STB
1	1	SB 8 bit data PB STB

The HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init() and HAL_MultiProcessor_Init() APIs follow respectively the UART asynchronous, UART Half duplex, LIN and Multi-Processor configuration procedures (details for the procedures are available in reference manual (RM0038)).

This section contains the following APIs:

- [HAL_UART_Init\(\)](#)
- [HAL_HalfDuplex_Init\(\)](#)
- [HAL_LIN_Init\(\)](#)
- [HAL_MultiProcessor_Init\(\)](#)
- [HAL_UART_DeInit\(\)](#)
- [HAL_UART_MspInit\(\)](#)
- [HAL_UART_MspDeInit\(\)](#)

43.3.3 IO operation functions

This subsection provides a set of functions allowing to manage the UART asynchronous and Half duplex data transfers.

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.

- Non blocking mode: The communication is performed using Interrupts or DMA, these APIs return the HAL status. The end of the data processing will be indicated through the dedicated UART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_UART_TxCpltCallback(), HAL_UART_RxCpltCallback() user callbacks will be executed respectively at the end of the transmit or receive process. The HAL_UART_ErrorCallback() user callback will be executed when a communication error is detected.
- 2. Blocking mode APIs are:
 - HAL_UART_Transmit()
 - HAL_UART_Receive()
- 3. Non Blocking mode APIs with Interrupt are:
 - HAL_UART_Transmit_IT()
 - HAL_UART_Receive_IT()
 - HAL_UART_IRQHandler()
- 4. Non Blocking mode functions with DMA are:
 - HAL_UART_Transmit_DMA()
 - HAL_UART_Receive_DMA()
 - HAL_UART_DMAPause()
 - HAL_UART_DMAResume()
 - HAL_UART_DMAStop()
- 5. A set of Transfer Complete Callbacks are provided in non blocking mode:
 - HAL_UART_TxHalfCpltCallback()
 - HAL_UART_TxCpltCallback()
 - HAL_UART_RxHalfCpltCallback()
 - HAL_UART_RxCpltCallback()
 - HAL_UART_ErrorCallback()



In the Half duplex communication, it is forbidden to run the transmit and receive process in parallel, the UART state HAL_UART_STATE_BUSY_TX_RX can't be useful.

This section contains the following APIs:

- [*HAL_UART_Transmit\(\)*](#)
- [*HAL_UART_Receive\(\)*](#)
- [*HAL_UART_Transmit_IT\(\)*](#)
- [*HAL_UART_Receive_IT\(\)*](#)
- [*HAL_UART_Transmit_DMA\(\)*](#)
- [*HAL_UART_Receive_DMA\(\)*](#)
- [*HAL_UART_DMAPause\(\)*](#)
- [*HAL_UART_DMAResume\(\)*](#)
- [*HAL_UART_DMAStop\(\)*](#)
- [*HAL_UART_IRQHandler\(\)*](#)
- [*HAL_UART_TxCpltCallback\(\)*](#)
- [*HAL_UART_TxHalfCpltCallback\(\)*](#)
- [*HAL_UART_RxCpltCallback\(\)*](#)
- [*HAL_UART_RxHalfCpltCallback\(\)*](#)
- [*HAL_UART_ErrorCallback\(\)*](#)

43.3.4 Peripheral Control functions

This subsection provides a set of functions allowing to control the UART:

- HAL_LIN_SendBreak() API can be helpful to transmit the break character.

- HAL_MultiProcessor_EnterMuteMode() API can be helpful to enter the UART in mute mode.
- HAL_MultiProcessor_ExitMuteMode() API can be helpful to exit the UART mute mode by software.
- HAL_HalfDuplex_EnableTransmitter() API to enable the UART transmitter and disables the UART receiver in Half Duplex mode
- HAL_HalfDuplex_EnableReceiver() API to enable the UART receiver and disables the UART transmitter in Half Duplex mode

This section contains the following APIs:

- [HAL_LIN_SendBreak\(\)](#)
- [HAL_MultiProcessor_EnterMuteMode\(\)](#)
- [HAL_MultiProcessor_ExitMuteMode\(\)](#)
- [HAL_HalfDuplex_EnableTransmitter\(\)](#)
- [HAL_HalfDuplex_EnableReceiver\(\)](#)

43.3.5 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of UART communication process, return Peripheral Errors occurred during communication process

- HAL_UART_GetState() API can be helpful to check in run-time the state of the UART peripheral.
- HAL_UART_GetError() check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- [HAL_UART_GetState\(\)](#)
- [HAL_UART_GetError\(\)](#)

43.3.6 HAL_UART_Init

Function Name	HAL_StatusTypeDef HAL_UART_Init (UART_HandleTypeDef * huart)
Function Description	Initializes the UART mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> • HAL status

43.3.7 HAL_HalfDuplex_Init

Function Name	HAL_StatusTypeDef HAL_HalfDuplex_Init (UART_HandleTypeDef * huart)
Function Description	Initializes the half-duplex mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> • HAL status

43.3.8 HAL_LIN_Init

Function Name	HAL_StatusTypeDef HAL_LIN_Init (UART_HandleTypeDef * huart, uint32_t BreakDetectLength)
Function Description	Initializes the LIN mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • BreakDetectLength: Specifies the LIN break detection length. This parameter can be one of the following values: UART_LINBREAKDETECTLENGTH_10B: 10-bit break detection UART_LINBREAKDETECTLENGTH_11B: 11-bit break detection
Return values	<ul style="list-style-type: none"> • HAL status

43.3.9 HAL_MultiProcessor_Init

Function Name	HAL_StatusTypeDef HAL_MultiProcessor_Init (UART_HandleTypeDef * huart, uint8_t Address, uint32_t WakeUpMethod)
Function Description	Initializes the Multi-Processor mode according to the specified parameters in the UART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • Address: UART node address • WakeUpMethod: specifies the UART wakeup method. This parameter can be one of the following values: UART_WAKEUPMETHOD_IDLELINE: Wakeup by an idle line detection UART_WAKEUPMETHOD_ADDRESSMARK: Wakeup by an address mark
Return values	<ul style="list-style-type: none"> • HAL status

43.3.10 HAL_UART_DeInit

Function Name	HAL_StatusTypeDef HAL_UART_DeInit (UART_HandleTypeDef * huart)
Function Description	DeInitializes the UART peripheral.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> • HAL status

43.3.11 HAL_UART_MspInit

Function Name	void HAL_UART_MspInit (UART_HandleTypeDef * huart)
Function Description	UART MSP Init.

Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> • None

43.3.12 HAL_UART_MspDeInit

Function Name	void HAL_UART_MspDeInit (UART_HandleTypeDef * huart)
Function Description	UART MSP DeInit.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> • None

43.3.13 HAL_UART_Transmit

Function Name	HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Sends an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • pData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

43.3.14 HAL_UART_Receive

Function Name	HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)
Function Description	Receives an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • pData: Pointer to data buffer • Size: Amount of data to be received • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

43.3.15 HAL_UART_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Sends an amount of data in non blocking mode.

Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

43.3.16 HAL_UART_Receive_IT

Function Name	HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Receives an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • pData: Pointer to data buffer • Size: Amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status

43.3.17 HAL_UART_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_UART_Transmit_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Sends an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • pData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status

43.3.18 HAL_UART_Receive_DMA

Function Name	HAL_StatusTypeDef HAL_UART_Receive_DMA (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
Function Description	Receives an amount of data in non blocking mode.
Parameters	<ul style="list-style-type: none"> • huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module. • pData: Pointer to data buffer • Size: Amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • When the UART parity is enabled (PCE = 1), the received data contain the parity bit (MSB position)

43.3.19 HAL_UART_DMAPause

Function Name	HAL_StatusTypeDef HAL_UART_DMAPause
---------------	--

(UART_HandleTypeDef * huart)

Function Description	Pauses the DMA Transfer.
Parameters	<ul style="list-style-type: none"> huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> HAL status

43.3.20 HAL_UART_DMAResume

Function Name	HAL_StatusTypeDef HAL_UART_DMAResume (UART_HandleTypeDef * huart)
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none"> huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> HAL status

43.3.21 HAL_UART_DMAStop

Function Name	HAL_StatusTypeDef HAL_UART_DMAStop (UART_HandleTypeDef * huart)
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none"> huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> HAL status

43.3.22 HAL_UART_IRQHandler

Function Name	void HAL_UART_IRQHandler (UART_HandleTypeDef * huart)
Function Description	This function handles UART interrupt request.
Parameters	<ul style="list-style-type: none"> huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> None

43.3.23 HAL_UART_TxCpltCallback

Function Name	void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none"> huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> None

43.3.24 HAL_UART_TxHalfCpltCallback

Function Name	void HAL_UART_TxHalfCpltCallback (UART_HandleTypeDef * huart)
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none">• None

43.3.25 HAL_UART_RxCpltCallback

Function Name	void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)
Function Description	Rx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none">• None

43.3.26 HAL_UART_RxHalfCpltCallback

Function Name	void HAL_UART_RxHalfCpltCallback (UART_HandleTypeDef * huart)
Function Description	Rx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none">• None

43.3.27 HAL_UART_ErrorCallback

Function Name	void HAL_UART_ErrorCallback (UART_HandleTypeDef * huart)
Function Description	UART error callbacks.
Parameters	<ul style="list-style-type: none">• huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none">• None

43.3.28 HAL_LIN_SendBreak

Function Name	HAL_StatusTypeDef HAL_LIN_SendBreak (UART_HandleTypeDef * huart)
Function Description	Transmits break characters.
Parameters	<ul style="list-style-type: none">• huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART

module.

Return values

- HAL status

43.3.29 HAL_MultiProcessor_EnterMuteMode

Function Name **HAL_StatusTypeDef HAL_MultiProcessor_EnterMuteMode (UART_HandleTypeDef * huart)**

Function Description Enters the UART in mute mode.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- HAL status

43.3.30 HAL_MultiProcessor_ExitMuteMode

Function Name **HAL_StatusTypeDef HAL_MultiProcessor_ExitMuteMode (UART_HandleTypeDef * huart)**

Function Description Exits the UART mute mode: wake up software.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- HAL status

43.3.31 HAL_HalfDuplex_EnableTransmitter

Function Name **HAL_StatusTypeDef HAL_HalfDuplex_EnableTransmitter (UART_HandleTypeDef * huart)**

Function Description Enables the UART transmitter and disables the UART receiver.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- HAL status

43.3.32 HAL_HalfDuplex_EnableReceiver

Function Name **HAL_StatusTypeDef HAL_HalfDuplex_EnableReceiver (UART_HandleTypeDef * huart)**

Function Description Enables the UART receiver and disables the UART transmitter.

Parameters

- **huart:** Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.

Return values

- HAL status

43.3.33 HAL_UART_GetState

Function Name **HAL_UART_StateTypeDef HAL_UART_GetState (UART_HandleTypeDef * huart)**

Function Description	Returns the UART state.
Parameters	<ul style="list-style-type: none"> huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART module.
Return values	<ul style="list-style-type: none"> HAL state

43.3.34 HAL_UART_GetError

Function Name	uint32_t HAL_UART_GetError (UART_HandleTypeDef * huart)
Function Description	Return the UART error code.
Parameters	<ul style="list-style-type: none"> huart: Pointer to a UART_HandleTypeDef structure that contains the configuration information for the specified UART.
Return values	<ul style="list-style-type: none"> UART Error Code

43.4 UART Firmware driver defines

43.4.1 UART

UART Error Codes

HAL_UART_ERROR_NONE	No error
HAL_UART_ERROR_PE	Parity error
HAL_UART_ERROR_NE	Noise error
HAL_UART_ERROR_FE	frame error
HAL_UART_ERROR_ORE	Overrun error
HAL_UART_ERROR_DMA	DMA transfer error

UART Exported Macros

`__HAL_UART_RESET_HANDLE_STATE`

Description:

- Reset UART handle state.

Parameters:

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

Return value:

- None

`__HAL_UART_FLUSH_DRREGISTER`

Description:

- Flush the UART DR register.

Parameters:

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).

__HAL_UART_GET_FLAG

values depending on device).

Description:

- Check whether the specified UART flag is set or not.

Parameters:

- **__HANDLE__**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- **__FLAG__**: specifies the flag to check. This parameter can be one of the following values:
 - **UART_FLAG_CTS**: CTS Change flag (not available for UART4 and UART5)
 - **UART_FLAG_LBD**: LIN Break detection flag
 - **UART_FLAG_TXE**: Transmit data register empty flag
 - **UART_FLAG_TC**: Transmission Complete flag
 - **UART_FLAG_RXNE**: Receive data register not empty flag
 - **UART_FLAG_IDLE**: Idle Line detection flag
 - **UART_FLAG_ORE**: OverRun Error flag
 - **UART_FLAG_NE**: Noise Error flag
 - **UART_FLAG_FE**: Framing Error flag
 - **UART_FLAG_PE**: Parity Error flag

Return value:

- The: new state of **__FLAG__** (TRUE or FALSE).

__HAL_UART_CLEAR_FLAG**Description:**

- Clear the specified UART pending flag.

Parameters:

- **__HANDLE__**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- **__FLAG__**: specifies the flag to check. This parameter can be any combination of the following values:

- UART_FLAG_CTS: CTS Change flag (not available for UART4 and UART5).
- UART_FLAG_LBD: LIN Break detection flag.
- UART_FLAG_TC: Transmission Complete flag.
- UART_FLAG_RXNE: Receive data register not empty flag.

Return value:

- None

Notes:

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART_SR register followed by a read operation to USART_DR register. RXNE flag can be also cleared by a read to the USART_DR register. TC flag can be also cleared by software sequence: a read operation to USART_SR register followed by a write operation to USART_DR register. TXE flag is cleared only by a write to the USART_DR register.

__HAL_UART_CLEAR_PEFLAG**Description:**

- Clear the UART PE pending flag.

Parameters:

- **__HANDLE__**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None

__HAL_UART_CLEAR_FEFLAG**Description:**

- Clear the UART FE pending flag.

Parameters:

- **__HANDLE__**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

__HAL_UART_CLEAR_NEFLAG

- None

Description:

- Clear the UART NE pending flag.

Parameters:

- **__HANDLE__**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None

__HAL_UART_CLEAR_OREFLAG**Description:**

- Clear the UART ORE pending flag.

Parameters:

- **__HANDLE__**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None

__HAL_UART_CLEAR_IDLEFLAG**Description:**

- Clear the UART IDLE pending flag.

Parameters:

- **__HANDLE__**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).

Return value:

- None

__HAL_UART_ENABLE_IT**Description:**

- Enable the specified UART interrupt.

Parameters:

- **__HANDLE__**: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,UART availability and x,y values depending on device).
- **__INTERRUPT__**: specifies the UART interrupt source to enable. This parameter can be one of the following values:

- UART_IT_CTS: CTS change interrupt
- UART_IT_LBD: LIN Break detection interrupt
- UART_IT_TXE: Transmit Data Register empty interrupt
- UART_IT_TC: Transmission complete interrupt
- UART_IT_RXNE: Receive Data register not empty interrupt
- UART_IT_IDLE: Idle line detection interrupt
- UART_IT_PE: Parity Error interrupt
- UART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

Description:

- Disable the specified UART interrupt.

Parameters:

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- `__INTERRUPT__`: specifies the UART interrupt source to disable. This parameter can be one of the following values:
 - UART_IT_CTS: CTS change interrupt
 - UART_IT_LBD: LIN Break detection interrupt
 - UART_IT_TXE: Transmit Data Register empty interrupt
 - UART_IT_TC: Transmission complete interrupt
 - UART_IT_RXNE: Receive Data register not empty interrupt
 - UART_IT_IDLE: Idle line detection interrupt
 - UART_IT_PE: Parity Error interrupt
 - UART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

`__HAL_UART_DISABLE_IT`

`__HAL_UART_GET_IT_SOURCE`

- None

Description:

- Check whether the specified UART interrupt has occurred or not.

Parameters:

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART, UART availability and x,y values depending on device).
- `__IT__`: specifies the UART interrupt source to check. This parameter can be one of the following values:
 - `UART_IT_CTS`: CTS change interrupt (not available for UART4 and UART5)
 - `UART_IT_LBD`: LIN Break detection interrupt
 - `UART_IT_TXE`: Transmit Data Register empty interrupt
 - `UART_IT_TC`: Transmission complete interrupt
 - `UART_IT_RXNE`: Receive Data register not empty interrupt
 - `UART_IT_IDLE`: Idle line detection interrupt
 - `UART_IT_ERR`: Error interrupt

Return value:

- The: new state of `__IT__` (TRUE or FALSE).

`__HAL_UART_ONE_BIT_SAMPLE_ENABLE`**Description:**

- macros to enables or disables the UART's one bit sampling method

Parameters:

- `__HANDLE__`: specifies the UART Handle. This parameter can be USARTx with x: 1, 2 or 3, or UARTy with y:4 or 5 to select the USART or UART peripheral (availability depending on device for UARTy).

Return value:

- None

`__HAL_UART_ONE_BIT_SAMPLE_DISABLE``__HAL_UART_HWCONTROL_CTS_ENABLE`**Description:**

- Enable CTS flow control This macro allows to enable CTS hardware flow control for a given UART instance,

without need to call

Parameters:

- `__HANDLE__`: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

Return value:

- None

Notes:

- As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : USART instance should have already been initialised (through call of `HAL_UART_Init()`) macro could only be called when corresponding UART instance is disabled (i.e `__HAL_UART_DISABLE(__HANDLE__)`) and should be followed by an Enable macro (i.e `__HAL_UART_ENABLE(__HANDLE__)`).

`__HAL_UART_HWCONTROL_CTS_DISABLE`**Description:**

- Disable CTS flow control This macro allows to disable CTS hardware flow control for a given UART instance, without need to call

Parameters:

- `__HANDLE__`: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

Return value:

- None

Notes:

- As macro is expected to be used for modifying CTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be

fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e __HAL_UART_ENABLE(__HANDLE__)).

`__HAL_UART_HWCONTROL_RTS_ENABLE`

Description:

- Enable RTS flow control This macro allows to enable RTS hardware flow control for a given UART instance, without need to call

Parameters:

- `__HANDLE__`: specifies the UART Handle. This parameter can be any USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

Return value:

- None

Notes:

- As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : UART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding UART instance is disabled (i.e __HAL_UART_DISABLE(__HANDLE__)) and should be followed by an Enable macro (i.e __HAL_UART_ENABLE(__HANDLE__)).

`__HAL_UART_HWCONTROL_RTS_DISABLE`

Description:

- Disable RTS flow control This macro allows to disable RTS hardware flow control for a given UART instance, without need to call

Parameters:

- `__HANDLE__`: specifies the UART Handle. This parameter can be any

USARTx (supporting the HW Flow control feature). It is used to select the USART peripheral (USART availability and x value depending on device).

Return value:

- None

Notes:

- As macro is expected to be used for modifying RTS Hw flow control feature activation, without need for USART instance Deinit/Init, following conditions for macro call should be fulfilled : USART instance should have already been initialised (through call of HAL_UART_Init())macro could only be called when corresponding USART instance is disabled (i.e `__HAL_UART_DISABLE(__HANDLE__)`) and should be followed by an Enable macro (i.e `__HAL_UART_ENABLE(__HANDLE__)`).

`__HAL_UART_ENABLE`

Description:

- Enable UART.

Parameters:

- `__HANDLE__`: specifies the UART Handle. UART Handle selects the USARTx or UARTy peripheral (USART,USART availability and x,y values depending on device).

Return value:

- None

`__HAL_UART_DISABLE`

Description:

- Disable UART UART Handle selects the USARTx or UARTy peripheral (USART,USART availability and x,y values depending on device).

Return value:

- None

UART Flags

`UART_FLAG_CTS`

`UART_FLAG_LBD`

`UART_FLAG_TXE`

`UART_FLAG_TC`

UART_FLAG_RXNE

UART_FLAG_IDLE

UART_FLAG_ORE

UART_FLAG_NE

UART_FLAG_FE

UART_FLAG_PE

UART Hardware Flow Control

UART_HWCONTROL_NONE

UART_HWCONTROL_RTS

UART_HWCONTROL_CTS

UART_HWCONTROL_RTS_CTS

UART Interrupt Definitions

UART_IT_PE

UART_IT_TXE

UART_IT_TC

UART_IT_RXNE

UART_IT_IDLE

UART_IT_LBD

UART_IT_CTS

UART_IT_ERR

UART LIN Break Detection Length

UART_LINBREAKDETECTLENGTH_10B

UART_LINBREAKDETECTLENGTH_11B

UART Transfer Mode

UART_MODE_RX

UART_MODE_TX

UART_MODE_TX_RX

UART Over Sampling

UART_OVERSAMPLING_16

UART_OVERSAMPLING_8

UART Parity

UART_PARITY_NONE

UART_PARITY_EVEN

UART_PARITY_ODD

UART Private Macros

UART_CR1_REG_INDEX

UART_CR2_REG_INDEX
UART_CR3_REG_INDEX
UART_DIV_SAMPLING16
UART_DIVMANT_SAMPLING16
UART_DIVFRAQ_SAMPLING16
UART_BRR_SAMPLING16
UART_DIV_SAMPLING8
UART_DIVMANT_SAMPLING8
UART_DIVFRAQ_SAMPLING8
UART_BRR_SAMPLING8
IS_UART_WORD_LENGTH
IS_UART_LIN_WORD_LENGTH
IS_UART_STOPBITS
IS_UART_PARITY
IS_UART_HARDWARE_FLOW_CONTROL
IS_UART_MODE
IS_UART_STATE
IS_UART_OVERSAMPLING
IS_UART_LIN_OVERSAMPLING
IS_UART_LIN_BREAK_DETECT_LENGTH
IS_UART_WAKEUPMETHOD
IS_UART_BAUDRATE

32 MHz) divided by the smallest
oversampling used on the USART (i.e. 8)
Return : TRUE or FALSE

IS_UART_ADDRESS

This parameter must be a number between
Min_Data = 0 and Max_Data = 15 Return :
TRUE or FALSE

UART_IT_MASK

UART State

UART_STATE_DISABLE

UART_STATE_ENABLE

UART Number of Stop Bits

UART_STOPBITS_1

UART_STOPBITS_2

UART Wakeup Functions

UART_WAKEUPMETHOD_IDLELINE

UART_WAKEUPMETHOD_ADDRESSMARK

UART Word Length

UART_WORDLENGTH_8B

UART_WORDLENGTH_9B

44 HAL USART Generic Driver

44.1 HAL USART Generic Driver

44.2 USART Firmware driver registers structures

44.2.1 USART_InitTypeDef

Data Fields

- *uint32_t BaudRate*
- *uint32_t WordLength*
- *uint32_t StopBits*
- *uint32_t Parity*
- *uint32_t Mode*
- *uint32_t CLKPolarity*
- *uint32_t CLKPhase*
- *uint32_t CLKLastBit*

Field Documentation

- ***uint32_t USART_InitTypeDef::BaudRate***
This member configures the Usart communication baud rate. The baud rate is computed using the following formula: $\text{IntegerDivider} = ((\text{PCLKx}) / (8 * (\text{USART_Init.BaudRate})))$ $\text{FractionalDivider} = ((\text{IntegerDivider} - ((\text{uint32_t}) \text{IntegerDivider})) * 8) + 0.5$
- ***uint32_t USART_InitTypeDef::WordLength***
Specifies the number of data bits transmitted or received in a frame. This parameter can be a value of [USART_Word_Length](#)
- ***uint32_t USART_InitTypeDef::StopBits***
Specifies the number of stop bits transmitted. This parameter can be a value of [USART_Stop_Bits](#)
- ***uint32_t USART_InitTypeDef::Parity***
Specifies the parity mode. This parameter can be a value of [USART_Parity](#)
Note:When parity is enabled, the computed parity is inserted at the MSB position of the transmitted data (9th bit when the word length is set to 9 data bits; 8th bit when the word length is set to 8 data bits).
- ***uint32_t USART_InitTypeDef::Mode***
Specifies whether the Receive or Transmit mode is enabled or disabled. This parameter can be a value of [USART_Mode](#)
- ***uint32_t USART_InitTypeDef::CLKPolarity***
Specifies the steady state of the serial clock. This parameter can be a value of [USART_Clock_Polarity](#)
- ***uint32_t USART_InitTypeDef::CLKPhase***
Specifies the clock transition on which the bit capture is made. This parameter can be a value of [USART_Clock_Phase](#)
- ***uint32_t USART_InitTypeDef::CLKLastBit***
Specifies whether the clock pulse corresponding to the last transmitted data bit (MSB) has to be output on the SCLK pin in synchronous mode. This parameter can be a value of [USART_Last_Bit](#)

44.2.2 USART_HandleTypeDef

Data Fields

- **USART_TypeDef * Instance**
- **USART_InitTypeDef Init**
- **uint8_t * pTxBuffPtr**
- **uint16_t TxXferSize**
- **__IO uint16_t TxXferCount**
- **uint8_t * pRxBuffPtr**
- **uint16_t RxXferSize**
- **__IO uint16_t RxXferCount**
- **DMA_HandleTypeDef * hdmatx**
- **DMA_HandleTypeDef * hdmarx**
- **HAL_LockTypeDef Lock**
- **__IO HAL_USART_StateTypeDef State**
- **__IO uint32_t ErrorCode**

Field Documentation

- **USART_TypeDef* USART_HandleTypeDef::Instance**
USART registers base address
- **USART_InitTypeDef USART_HandleTypeDef::Init**
Usart communication parameters
- **uint8_t* USART_HandleTypeDef::pTxBuffPtr**
Pointer to Usart Tx transfer Buffer
- **uint16_t USART_HandleTypeDef::TxXferSize**
Usart Tx Transfer size
- **__IO uint16_t USART_HandleTypeDef::TxXferCount**
Usart Tx Transfer Counter
- **uint8_t* USART_HandleTypeDef::pRxBuffPtr**
Pointer to Usart Rx transfer Buffer
- **uint16_t USART_HandleTypeDef::RxXferSize**
Usart Rx Transfer size
- **__IO uint16_t USART_HandleTypeDef::RxXferCount**
Usart Rx Transfer Counter
- **DMA_HandleTypeDef* USART_HandleTypeDef::hdmatx**
Usart Tx DMA Handle parameters
- **DMA_HandleTypeDef* USART_HandleTypeDef::hdmarx**
Usart Rx DMA Handle parameters
- **HAL_LockTypeDef USART_HandleTypeDef::Lock**
Locking object
- **__IO HAL_USART_StateTypeDef USART_HandleTypeDef::State**
Usart communication state
- **__IO uint32_t USART_HandleTypeDef::ErrorCode**
USART Error code

44.3 USART Firmware driver API description

44.3.1 How to use this driver

The USART HAL driver can be used as follows:

1. Declare a USART_HandleTypeDef handle structure.
2. Initialize the USART low level resources by implementing the HAL_USART_MspInit() API:
 - a. Enable the USARTx interface clock.
 - b. USART pins configuration:
 - Enable the clock for the USART GPIOs.
 - Configure the USART pins as alternate function pull-up.
 - c. NVIC configuration if you need to use interrupt process (HAL_USART_Transmit_IT(), HAL_USART_Receive_IT() and HAL_USART_TransmitReceive_IT() APIs):
 - Configure the USARTx interrupt priority.
 - Enable the NVIC USART IRQ handle.
 - d. DMA Configuration if you need to use DMA process (HAL_USART_Transmit_DMA() HAL_USART_Receive_DMA() and HAL_USART_TransmitReceive_DMA() APIs):
 - Declare a DMA handle structure for the Tx/Rx channel.
 - Enable the DMAx interface clock.
 - Configure the declared DMA handle structure with the required Tx/Rx parameters.
 - Configure the DMA Tx/Rx channel.
 - Associate the initialized DMA handle to the USART DMA Tx/Rx handle.
 - Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx channel.
 - Configure the USARTx interrupt priority and enable the NVIC USART IRQ handle (used for last byte sending completion detection in DMA non circular mode)
3. Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware flow control and Mode(Receiver/Transmitter) in the husart Init structure.
4. Initialize the USART registers by calling the HAL_USART_Init() API:
 - These APIs configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_USART_MspInit(&husart) API. The specific USART interrupts (Transmission complete interrupt, RXNE interrupt and Error Interrupts) will be managed using the macros __HAL_USART_ENABLE_IT() and __HAL_USART_DISABLE_IT() inside the transmit and receive process.
5. Three operation modes are available within this driver :

Polling mode IO operation

- Send an amount of data in blocking mode using HAL_USART_Transmit()
- Receive an amount of data in blocking mode using HAL_USART_Receive()

Interrupt mode IO operation

- Send an amount of data in non blocking mode using HAL_USART_Transmit_IT()

- At transmission end of transfer HAL_USART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxCpltCallback
- Receive an amount of data in non blocking mode using HAL_USART_Receive_IT()
- At reception end of transfer HAL_USART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxCpltCallback
- In case of transfer Error, HAL_USART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_USART_ErrorCallback

DMA mode IO operation

- Send an amount of data in non blocking mode (DMA) using HAL_USART_Transmit_DMA()
- At transmission end of half transfer HAL_USART_TxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxHalfCpltCallback
- At transmission end of transfer HAL_USART_TxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_TxCpltCallback
- Receive an amount of data in non blocking mode (DMA) using HAL_USART_Receive_DMA()
- At reception end of half transfer HAL_USART_RxHalfCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxHalfCpltCallback
- At reception end of transfer HAL_USART_RxCpltCallback is executed and user can add his own code by customization of function pointer HAL_USART_RxCpltCallback
- In case of transfer Error, HAL_USART_ErrorCallback() function is executed and user can add his own code by customization of function pointer HAL_USART_ErrorCallback
- Pause the DMA Transfer using HAL_USART_DMAPause()
- Resume the DMA Transfer using HAL_USART_DMAResume()
- Stop the DMA Transfer using HAL_USART_DMAStop()

USART HAL driver macros list

Below the list of most used macros in USART HAL driver.

- __HAL_USART_ENABLE: Enable the USART peripheral
- __HAL_USART_DISABLE: Disable the USART peripheral
- __HAL_USART_GET_FLAG : Check whether the specified USART flag is set or not
- __HAL_USART_CLEAR_FLAG : Clear the specified USART pending flag
- __HAL_USART_ENABLE_IT: Enable the specified USART interrupt
- __HAL_USART_DISABLE_IT: Disable the specified USART interrupt
- __HAL_USART_GET_IT_SOURCE: Check whether the specified USART interrupt has occurred or not



You can refer to the USART HAL driver header file for more useful macros

44.3.2 Initialization and Configuration functions

This subsection provides a set of functions allowing to initialize the USART in asynchronous and in synchronous modes.

- For the asynchronous mode only these parameters can be configured:
 - Baud Rate
 - Word Length
 - Stop Bit
 - Parity: If the parity is enabled, then the MSB bit of the data written in the data register is transmitted but is changed by the parity bit. Depending on the frame length defined by the M bit (8-bits or 9-bits), the possible USART frame formats are as listed in [Table 19: "OPAMPs inverting/non-inverting inputs for STM32L1 devices"](#).
 - USART polarity
 - USART phase
 - USART LastBit
 - Receiver/transmitter modes

Table 25: USART frame formats

M bit	PCE bit	USART frame
0	0	SB 8 bit data STB
0	1	SB 7 bit data PB STB
1	0	SB 9 bit data STB
1	1	SB 8 bit data PB STB

The HAL_USART_Init() function follows the USART synchronous configuration procedure (details for the procedure are available in reference manual (RM0038)).

This section contains the following APIs:

- [HAL_USART_Init\(\)](#)
- [HAL_USART_DeInit\(\)](#)
- [HAL_USART_MspInit\(\)](#)
- [HAL_USART_MspDeInit\(\)](#)

44.3.3 IO operation functions

This subsection provides a set of functions allowing to manage the USART synchronous data transfers.

The USART supports master mode only: it cannot receive or send data related to an input clock (SCLK is always an output).

1. There are two modes of transfer:
 - Blocking mode: The communication is performed in polling mode. The HAL status of all data processing is returned by the same function after finishing transfer.
 - No-Blocking mode: The communication is performed using Interrupts or DMA. These APIs return the HAL status. The end of the data processing will be indicated through the dedicated USART IRQ when using Interrupt mode or the DMA IRQ when using DMA mode. The HAL_USART_TxCpltCallback(), HAL_USART_RxCpltCallback() and HAL_USART_TxRxCpltCallback() user callbacks will be executed respectively at the end of the transmit or Receive

process The HAL_USART_ErrorCallback() user callback will be executed when a communication error is detected

2. Blocking mode APIs are :
 - HAL_USART_Transmit() in simplex mode
 - HAL_USART_Receive() in full duplex receive only
 - HAL_USART_TransmitReceive() in full duplex mode
3. Non Blocking mode APIs with Interrupt are :
 - HAL_USART_Transmit_IT() in simplex mode
 - HAL_USART_Receive_IT() in full duplex receive only
 - HAL_USART_TransmitReceive_IT() in full duplex mode
 - HAL_USART_IRQHandler()
4. Non Blocking mode functions with DMA are :
 - HAL_USART_Transmit_DMA() in simplex mode
 - HAL_USART_Receive_DMA() in full duplex receive only
 - HAL_USART_TransmitReceive_DMA() in full duplex mode
 - HAL_USART_DMAPause()
 - HAL_USART_DMAResume()
 - HAL_USART_DMAStop()
5. A set of Transfer Complete Callbacks are provided in non Blocking mode:
 - HAL_USART_TxHalfCpltCallback()
 - HAL_USART_TxCpltCallback()
 - HAL_USART_RxHalfCpltCallback()
 - HAL_USART_RxCpltCallback()
 - HAL_USART_ErrorCallback()
 - HAL_USART_TxRxCpltCallback()

This section contains the following APIs:

- [*HAL_USART_Transmit\(\)*](#)
- [*HAL_USART_Receive\(\)*](#)
- [*HAL_USART_TransmitReceive\(\)*](#)
- [*HAL_USART_Transmit_IT\(\)*](#)
- [*HAL_USART_Receive_IT\(\)*](#)
- [*HAL_USART_TransmitReceive_IT\(\)*](#)
- [*HAL_USART_Transmit_DMA\(\)*](#)
- [*HAL_USART_Receive_DMA\(\)*](#)
- [*HAL_USART_TransmitReceive_DMA\(\)*](#)
- [*HAL_USART_DMAPause\(\)*](#)
- [*HAL_USART_DMAResume\(\)*](#)
- [*HAL_USART_DMAStop\(\)*](#)
- [*HAL_USART_IRQHandler\(\)*](#)
- [*HAL_USART_TxCpltCallback\(\)*](#)
- [*HAL_USART_TxHalfCpltCallback\(\)*](#)
- [*HAL_USART_RxCpltCallback\(\)*](#)
- [*HAL_USART_RxHalfCpltCallback\(\)*](#)
- [*HAL_USART_TxRxCpltCallback\(\)*](#)
- [*HAL_USART_ErrorCallback\(\)*](#)

44.3.4 Peripheral State and Errors functions

This subsection provides a set of functions allowing to return the State of USART communication process, return Peripheral Errors occurred during communication process

- HAL_USART_GetState() API can be helpful to check in run-time the state of the USART peripheral.

- HAL_USART_GetError() check in run-time errors that could be occurred during communication.

This section contains the following APIs:

- [HAL_USART_GetState\(\)](#)
- [HAL_USART_GetError\(\)](#)

44.3.5 HAL_USART_Init

Function Name	HAL_StatusTypeDef HAL_USART_Init (USART_HandleTypeDef * husart)
Function Description	Initializes the USART mode according to the specified parameters in the USART_InitTypeDef and create the associated handle.
Parameters	<ul style="list-style-type: none">• husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none">• HAL status

44.3.6 HAL_USART_DeInit

Function Name	HAL_StatusTypeDef HAL_USART_DeInit (USART_HandleTypeDef * husart)
Function Description	DeInitializes the USART peripheral.
Parameters	<ul style="list-style-type: none">• husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none">• HAL status

44.3.7 HAL_USART_MspInit

Function Name	void HAL_USART_MspInit (USART_HandleTypeDef * husart)
Function Description	USART MSP Init.
Parameters	<ul style="list-style-type: none">• husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none">• None

44.3.8 HAL_USART_MspDeInit

Function Name	void HAL_USART_MspDeInit (USART_HandleTypeDef * husart)
Function Description	USART MSP DeInit.
Parameters	<ul style="list-style-type: none">• husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none">• None

44.3.9 HAL_USART_Transmit

Function Name	HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)
Function Description	Simplex Send an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • pTxData: Pointer to data buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

44.3.10 HAL_USART_Receive

Function Name	HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Full-Duplex Receive an amount of data in blocking mode.
Parameters	<ul style="list-style-type: none"> • husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • pRxData: Pointer to data buffer • Size: Amount of data to be received • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

44.3.11 HAL_USART_TransmitReceive

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)
Function Description	Full-Duplex Send receive an amount of data in full-duplex mode (blocking mode).
Parameters	<ul style="list-style-type: none"> • husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • pTxData: Pointer to data transmitted buffer • pRxData: Pointer to data received buffer • Size: Amount of data to be sent • Timeout: Timeout duration
Return values	<ul style="list-style-type: none"> • HAL status

44.3.12 HAL_USART_Transmit_IT

Function Name	HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)
Function Description	Simplex Send an amount of data in non-blocking mode.

Parameters	<ul style="list-style-type: none"> • husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • pTxData: Pointer to data buffer • Size: Amount of data to be sent
Return values	<ul style="list-style-type: none"> • HAL status
Notes	<ul style="list-style-type: none"> • The USART errors are not managed to avoid the overrun error.

44.3.13 HAL_USART_Receive_IT

Function Name	HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)
Function Description	Simplex Receive an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> • husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • pRxData: Pointer to data buffer • Size: Amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status

44.3.14 HAL_USART_TransmitReceive_IT

Function Name	HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)
Function Description	Full-Duplex Send receive an amount of data in full-duplex mode (non-blocking).
Parameters	<ul style="list-style-type: none"> • husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • pTxData: Pointer to data transmitted buffer • pRxData: Pointer to data received buffer • Size: Amount of data to be received
Return values	<ul style="list-style-type: none"> • HAL status

44.3.15 HAL_USART_Transmit_DMA

Function Name	HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)
Function Description	Simplex Send an amount of data in non-blocking mode.
Parameters	<ul style="list-style-type: none"> • husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module. • pTxData: Pointer to data buffer • Size: Amount of data to be sent

Return values

- HAL status

44.3.16 HAL_USART_Receive_DMA

Function Name `HAL_StatusTypeDef HAL_USART_Receive_DMA(USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)`

Function Description Full-Duplex Receive an amount of data in non-blocking mode.

Parameters

- **husart**: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pRxData**: Pointer to data buffer
- **Size**: Amount of data to be received

Return values

- HAL status

Notes

- The USART DMA transmit channel must be configured in order to generate the clock for the slave.
- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

44.3.17 HAL_USART_TransmitReceive_DMA

Function Name `HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA(USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size)`

Function Description Full-Duplex Transmit Receive an amount of data in non-blocking mode.

Parameters

- **husart**: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
- **pTxData**: Pointer to data transmitted buffer
- **pRxData**: Pointer to data received buffer
- **Size**: Amount of data to be received

Return values

- HAL status

Notes

- When the USART parity is enabled (PCE = 1) the data received contain the parity bit.

44.3.18 HAL_USART_DMAPause

Function Name `HAL_StatusTypeDef HAL_USART_DMAPause(USART_HandleTypeDef * husart)`

Function Description Pauses the DMA Transfer.

Parameters

- **husart**: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- HAL status

44.3.19 HAL_USART_DMAResume

Function Name	HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart)
Function Description	Resumes the DMA Transfer.
Parameters	<ul style="list-style-type: none">• husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none">• HAL status

44.3.20 HAL_USART_DMAStop

Function Name	HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart)
Function Description	Stops the DMA Transfer.
Parameters	<ul style="list-style-type: none">• husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none">• HAL status

44.3.21 HAL_USART_IRQHandler

Function Name	void HAL_USART_IRQHandler (USART_HandleTypeDef * husart)
Function Description	This function handles USART interrupt request.
Parameters	<ul style="list-style-type: none">• husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none">• None

44.3.22 HAL_USART_TxCpltCallback

Function Name	void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none">• None

44.3.23 HAL_USART_TxHalfCpltCallback

Function Name	void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart)
Function Description	Tx Half Transfer completed callbacks.
Parameters	<ul style="list-style-type: none">• husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- None

44.3.24 HAL_USART_RxCpltCallback

Function Name **void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart)**

Function Description Rx Transfer completed callbacks.

Parameters

- **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- None

44.3.25 HAL_USART_RxHalfCpltCallback

Function Name **void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart)**

Function Description Rx Half Transfer completed callbacks.

Parameters

- **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- None

44.3.26 HAL_USART_TxRxCpltCallback

Function Name **void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart)**

Function Description Tx/Rx Transfers completed callback for the non-blocking process.

Parameters

- **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- None

44.3.27 HAL_USART_ErrorCallback

Function Name **void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart)**

Function Description USART error callbacks.

Parameters

- **husart:** Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.

Return values

- None

44.3.28 HAL_USART_GetState

Function Name **HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)**

Function Description Returns the USART state.

Parameters	<ul style="list-style-type: none"> husart: Pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART module.
Return values	<ul style="list-style-type: none"> HAL state

44.3.29 HAL_USART_GetError

Function Name	uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)
Function Description	Return the USART error code.
Parameters	<ul style="list-style-type: none"> husart: : pointer to a USART_HandleTypeDef structure that contains the configuration information for the specified USART.
Return values	<ul style="list-style-type: none"> USART Error Code

44.4 USART Firmware driver defines

44.4.1 USART

USART Clock

USART_CLOCK_DISABLE

USART_CLOCK_ENABLE

USART Clock Phase

USART_PHASE_1EDGE

USART_PHASE_2EDGE

USART Clock Polarity

USART_POLARITY_LOW

USART_POLARITY_HIGH

USART Error Codes

HAL_USART_ERROR_NONE	No error
HAL_USART_ERROR_PE	Parity error
HAL_USART_ERROR_NE	Noise error
HAL_USART_ERROR_FE	frame error
HAL_USART_ERROR_ORE	Overrun error
HAL_USART_ERROR_DMA	DMA transfer error

USART Exported Macros

__HAL_USART_RESET_HANDLE_STATE

Description:

- Reset USART handle state.

Parameters:

- __HANDLE__**: specifies the USART Handle. USART Handle selects the USARTx peripheral

(USART availability and x value depending on device).

Return value:

- None

Description:

- Check whether the specified USART flag is set or not.

Parameters:

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `USART_FLAG_TXE`: Transmit data register empty flag
 - `USART_FLAG_TC`: Transmission Complete flag
 - `USART_FLAG_RXNE`: Receive data register not empty flag
 - `USART_FLAG_IDLE`: Idle Line detection flag
 - `USART_FLAG_ORE`: OverRun Error flag
 - `USART_FLAG_NE`: Noise Error flag
 - `USART_FLAG_FE`: Framing Error flag
 - `USART_FLAG_PE`: Parity Error flag

Return value:

- The: new state of `__FLAG__` (TRUE or FALSE).

Description:

- Clear the specified USART pending flags.

Parameters:

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__FLAG__`: specifies the flag to check. This parameter can be any combination of the following values:

`__HAL_USART_GET_FLAG`

`__HAL_USART_CLEAR_FLAG`

- USART_FLAG_TC:
Transmission Complete flag.
- USART_FLAG_RXNE:
Receive data register not empty flag.

Return value:

- None

Notes:

- PE (Parity error), FE (Framing error), NE (Noise error), ORE (OverRun error) and IDLE (Idle line detected) flags are cleared by software sequence: a read operation to USART_SR register followed by a read operation to USART_DR register. RXNE flag can be also cleared by a read to the USART_DR register. TC flag can be also cleared by software sequence: a read operation to USART_SR register followed by a write operation to USART_DR register. TXE flag is cleared only by a write to the USART_DR register.

__HAL_USART_CLEAR_PEFLAG**Description:**

- Clear the USART PE pending flag.

Parameters:

- **__HANDLE__**: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

__HAL_USART_CLEAR_FEFLAG**Description:**

- Clear the USART FE pending flag.

Parameters:

- **__HANDLE__**: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

__HAL_USART_CLEAR_NEFLAG**Description:**

- Clear the USART NE pending flag.

Parameters:

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Clear the USART ORE pending flag.

Parameters:

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Clear the USART IDLE pending flag.

Parameters:

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Enable the specified Usart interrupts.

Parameters:

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__INTERRUPT__`: specifies the USART interrupt source to enable. This parameter can be one of the following values:

`__HAL_USART_CLEAR_OREFLAG`

`__HAL_USART_CLEAR_IDLEFLAG`

`__HAL_USART_ENABLE_IT`

- USART_IT_TXE: Transmit Data Register empty interrupt
- USART_IT_TC: Transmission complete interrupt
- USART_IT_RXNE: Receive Data register not empty interrupt
- USART_IT_IDLE: Idle line detection interrupt
- USART_IT_PE: Parity Error interrupt
- USART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

Description:

- Disable the specified Usart interrupts.

Parameters:

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__INTERRUPT__`: specifies the USART interrupt source to disable. This parameter can be one of the following values:
 - USART_IT_TXE: Transmit Data Register empty interrupt
 - USART_IT_TC: Transmission complete interrupt
 - USART_IT_RXNE: Receive Data register not empty interrupt
 - USART_IT_IDLE: Idle line detection interrupt
 - USART_IT_PE: Parity Error interrupt
 - USART_IT_ERR: Error interrupt(Frame error, noise error, overrun error)

Return value:

- None

Description:

- Check whether the specified Usart interrupt has occurred or not.

`__HAL_USART_DISABLE_IT`

`__HAL_USART_GET_IT_SOURCE`

Parameters:

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).
- `__IT__`: specifies the USART interrupt source to check. This parameter can be one of the following values:
 - `USART_IT_TXE`: Transmit Data Register empty interrupt
 - `USART_IT_TC`: Transmission complete interrupt
 - `USART_IT_RXNE`: Receive Data register not empty interrupt
 - `USART_IT_IDLE`: Idle line detection interrupt
 - `USART_IT_ERR`: Error interrupt
 - `USART_IT_PE`: Parity Error interrupt

Return value:

- The: new state of `__IT__` (TRUE or FALSE).

`__HAL_USART_ONE_BIT_SAMPLE_ENABLE`**Description:**

- Enables the USART one bit sample method.

Parameters:

- `__HANDLE__`: specifies the USART Handle.

Return value:

- None

`__HAL_USART_ONE_BIT_SAMPLE_DISABLE`**Description:**

- Disables the UART one bit sample method.

Parameters:

- `__HANDLE__`: specifies the UART Handle.

Return value:

- None

`__HAL_USART_ENABLE`**Description:**

- Enable USART.

Parameters:

__HAL_USART_DISABLE

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

Description:

- Disable USART.

Parameters:

- `__HANDLE__`: specifies the USART Handle. USART Handle selects the USARTx peripheral (USART availability and x value depending on device).

Return value:

- None

USART Flags

USART_FLAG_CTS

USART_FLAG_LBD

USART_FLAG_TXE

USART_FLAG_TC

USART_FLAG_RXNE

USART_FLAG_IDLE

USART_FLAG_ORE

USART_FLAG_NE

USART_FLAG_FE

USART_FLAG_PE

USART Interrupts Definition

USART_IT_PE

USART_IT_TXE

USART_IT_TC

USART_IT_RXNE

USART_IT_IDLE

USART_IT_LBD

USART_IT_CTS

USART_IT_ERR

USART Last Bit

USART_LASTBIT_DISABLE

USART_LASTBIT_ENABLE

USART Mode

USART_MODE_RX

USART_MODE_TX

USART_MODE_TX_RX

USART NACK State

USART_NACK_ENABLE

USART_NACK_DISABLE

USART Parity

USART_PARITY_NONE

USART_PARITY_EVEN

USART_PARITY_ODD

USART Private Constants

DUMMY_DATA

USART Private Macros

USART_CR1_REG_INDEX

USART_CR2_REG_INDEX

USART_CR3_REG_INDEX

USART_DIV

USART_DIVMANT

USART_DIVFRAQ

USART_BRR

IS_USART_BAUDRATE 32 MHz) divided by the smallest oversampling used on the
USART (i.e. 8) return : TRUE or FALSE

IS_USART_WORD_LENGTH

IS_USART_STOPBITS

IS_USART_PARITY

IS_USART_MODE

IS_USART_CLOCK

IS_USART_POLARITY

IS_USART_PHASE

IS_USART_LASTBIT

IS_USART_NACK_STATE

USART_IT_MASK

USART Number of Stop Bits

USART_STOPBITS_1

USART_STOPBITS_0_5

USART_STOPBITS_2

USART_STOPBITS_1_5

USART Word Length

USART_WORDLENGTH_8B

USART_WORDLENGTH_9B

45 HAL WWDG Generic Driver

45.1 HAL WWDG Generic Driver

45.2 WWDG Firmware driver registers structures

45.2.1 WWDG_InitTypeDef

Data Fields

- *uint32_t Prescaler*
- *uint32_t Window*
- *uint32_t Counter*

Field Documentation

- *uint32_t WWDG_InitTypeDef::Prescaler*
Specifies the prescaler value of the WWDG. This parameter can be a value of [WWDG_Prescaler](#)
- *uint32_t WWDG_InitTypeDef::Window*
Specifies the WWDG window value to be compared to the downcounter. This parameter must be a number lower than Max_Data = 0x80
- *uint32_t WWDG_InitTypeDef::Counter*
Specifies the WWDG free-running downcounter value. This parameter must be a number between Min_Data = 0x40 and Max_Data = 0x7F

45.2.2 WWDG_HandleTypeDef

Data Fields

- *WWDG_TypeDef * Instance*
- *WWDG_InitTypeDef Init*
- *HAL_LockTypeDef Lock*
- *__IO HAL_WWDG_StateTypeDef State*

Field Documentation

- *WWDG_TypeDef* WWDG_HandleTypeDef::Instance*
Register base address
- *WWDG_InitTypeDef WWDG_HandleTypeDef::Init*
WWDG required parameters
- *HAL_LockTypeDef WWDG_HandleTypeDef::Lock*
WWDG locking object
- *__IO HAL_WWDG_StateTypeDef WWDG_HandleTypeDef::State*
WWDG communication state

45.3 WWDG Firmware driver API description

45.3.1 WWDG specific features

Once enabled the WWDG generates a system reset on expiry of a programmed time period, unless the program refreshes the counter (downcounter) before reaching 0x3F value (i.e. a reset is generated when the counter value rolls over from 0x40 to 0x3F).

- An MCU reset is also generated if the counter value is refreshed before the counter has reached the refresh window value. This implies that the counter must be refreshed in a limited window.
- Once enabled the WWDG cannot be disabled except by a system reset.
- WWDGRST flag in RCC_CSR register can be used to inform when a WWDG reset occurs.
- The WWDG counter input clock is derived from the APB clock divided by a programmable prescaler.
- $\text{WWDG clock (Hz)} = \text{PCLK1} / (4096 * \text{Prescaler})$
- $\text{WWDG timeout (mS)} = 1000 * \text{Counter} / \text{WWDG clock}$
- WWDG Counter refresh is allowed between the following limits :
 - $\text{min time (mS)} = 1000 * (\text{Counter} - \text{Window}) / \text{WWDG clock}$
 - $\text{max time (mS)} = 1000 * (\text{Counter} - 0x40) / \text{WWDG clock}$
- Min-max timeout value at @32MHz (PCLK1): ~128us / ~65.6ms.

45.3.2 How to use this driver

- Enable WWDG APB1 clock using `__HAL_RCC_WWDG_CLK_ENABLE()`.
- Set the WWDG prescaler, refresh window and counter value using `HAL_WWDG_Init()` function.
- Start the WWDG using `HAL_WWDG_Start()` function. When the WWDG is enabled the counter value should be configured to a value greater than 0x40 to prevent generating an immediate reset.
- Optionally you can enable the Early Wakeup Interrupt (EWI) which is generated when the counter reaches 0x40, and then start the WWDG using `HAL_WWDG_Start_IT()`. At EWI `HAL_WWDG_WakeupCallback` is executed and user can add his own code by customization of function pointer `HAL_WWDG_WakeupCallback`. Once enabled, EWI interrupt cannot be disabled except by a system reset.
- Then the application program must refresh the WWDG counter at regular intervals during normal operation to prevent an MCU reset, using `HAL_WWDG_Refresh()` function. This operation must occur only when the counter is lower than the refresh window value already programmed.

WWDG HAL driver macros list

Below the list of most used macros in WWDG HAL driver.

- `__HAL_WWDG_ENABLE`: Enable the WWDG peripheral
- `__HAL_WWDG_GET_FLAG`: Get the selected WWDG's flag status
- `__HAL_WWDG_CLEAR_FLAG`: Clear the WWDG's pending flags
- `__HAL_WWDG_ENABLE_IT`: Enables the WWDG early wakeup interrupt

45.3.3 Initialization and de-initialization functions

This section provides functions allowing to:

- Initialize the WWDG according to the specified parameters in the WWDG_InitTypeDef and create the associated handle
- DeInitialize the WWDG peripheral
- Initialize the WWDG MSP
- DeInitialize the WWDG MSP

This section contains the following APIs:

- [HAL_WWDG_Init\(\)](#)
- [HAL_WWDG_DeInit\(\)](#)
- [HAL_WWDG_MspInit\(\)](#)
- [HAL_WWDG_MspDeInit\(\)](#)
- [HAL_WWDG_WakeupCallback\(\)](#)

45.3.4 IO operation functions

This section provides functions allowing to:

- Start the WWDG.
- Refresh the WWDG.
- Handle WWDG interrupt request.

This section contains the following APIs:

- [HAL_WWDG_Start\(\)](#)
- [HAL_WWDG_Start_IT\(\)](#)
- [HAL_WWDG_Refresh\(\)](#)
- [HAL_WWDG_IRQHandler\(\)](#)
- [HAL_WWDG_WakeupCallback\(\)](#)

45.3.5 Peripheral State functions

This subsection permits to get in run-time the status of the peripheral and the data flow.

This section contains the following APIs:

- [HAL_WWDG_GetState\(\)](#)

45.3.6 HAL_WWDG_Init

Function Name	HAL_StatusTypeDef HAL_WWDG_Init(WWDG_HandleTypeDef * hwwdg)
Function Description	Initializes the WWDG according to the specified parameters in the WWDG_InitTypeDef and creates the associated handle.
Parameters	<ul style="list-style-type: none"> • hwwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> • HAL status

45.3.7 HAL_WWDG_DeInit

Function Name	HAL_StatusTypeDef HAL_WWDG_DeInit(WWDG_HandleTypeDef * hwwdg)
Function Description	DeInitializes the WWDG peripheral.
Parameters	<ul style="list-style-type: none"> • hwwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified

WWDG module.

Return values

- HAL status

45.3.8 HAL_WWDG_MspInit

Function Name **void HAL_WWDG_MspInit (WWDG_HandleTypeDef * hwwdg)**

Function Description Initializes the WWDG MSP.

Parameters

- **hwwdg**: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- None

45.3.9 HAL_WWDG_MspDeInit

Function Name **void HAL_WWDG_MspDeInit (WWDG_HandleTypeDef * hwwdg)**

Function Description DeInitializes the WWDG MSP.

Parameters

- **hwwdg**: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- None

45.3.10 HAL_WWDG_WakeupCallback

Function Name **void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwwdg)**

Function Description Early Wakeup WWDG callback.

Parameters

- **hwwdg**: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- None

45.3.11 HAL_WWDG_Start

Function Name **HAL_StatusTypeDef HAL_WWDG_Start (WWDG_HandleTypeDef * hwwdg)**

Function Description Starts the WWDG.

Parameters

- **hwwdg**: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.

Return values

- HAL status

45.3.12 HAL_WWDG_Start_IT

Function Name **HAL_StatusTypeDef HAL_WWDG_Start_IT (WWDG_HandleTypeDef * hwwdg)**

Function Description Starts the WWDG with interrupt enabled.

Parameters	<ul style="list-style-type: none"> hwwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> HAL status

45.3.13 HAL_WWDG_Refresh

Function Name	HAL_StatusTypeDef HAL_WWDG_Refresh (WWDG_HandleTypeDef * hwwdg, uint32_t Counter)
Function Description	Refreshes the WWDG.
Parameters	<ul style="list-style-type: none"> hwwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module. Counter: value of counter to put in WWDG counter
Return values	<ul style="list-style-type: none"> HAL status

45.3.14 HAL_WWDG_IRQHandler

Function Name	void HAL_WWDG_IRQHandler (WWDG_HandleTypeDef * hwwdg)
Function Description	Handles WWDG interrupt request.
Parameters	<ul style="list-style-type: none"> hwwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> None
Notes	<ul style="list-style-type: none"> The Early Wakeup Interrupt (EWI) can be used if specific safety operations or data logging must be performed before the actual reset is generated. The EWI interrupt is enabled when calling HAL_WWDG_Start_IT function. When the downcounter reaches the value 0x40, and EWI interrupt is generated and the corresponding Interrupt Service Routine (ISR) can be used to trigger specific actions (such as communications or data logging), before resetting the device.

45.3.15 HAL_WWDG_WakeupCallback

Function Name	void HAL_WWDG_WakeupCallback (WWDG_HandleTypeDef * hwwdg)
Function Description	Early Wakeup WWDG callback.
Parameters	<ul style="list-style-type: none"> hwwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> None

45.3.16 HAL_WWDG_GetState

Function Name	HAL_WWDG_StateTypeDef HAL_WWDG_GetState (WWDG_HandleTypeDef * hwwdg)
---------------	---

Function Description	Returns the WWDG state.
Parameters	<ul style="list-style-type: none"> hwwdg: pointer to a WWDG_HandleTypeDef structure that contains the configuration information for the specified WWDG module.
Return values	<ul style="list-style-type: none"> HAL state

45.4 WWDG Firmware driver defines

45.4.1 WWDG

WWDG Counter

IS_WWDG_COUNTER

WWDG Exported Macros

__HAL_WWDG_RESET_HANDLE_STATE **Description:**

- Reset WWDG handle state.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_ENABLE

Description:

- Enables the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

__HAL_WWDG_DISABLE

Description:

- Disables the WWDG peripheral.

Parameters:

- __HANDLE__: WWDG handle

Return value:

- None

Notes:

- WARNING: This is a dummy macro for HAL code alignment. Once enable, WWDG Peripheral cannot be disabled except by a system reset.

__HAL_WWDG_ENABLE_IT

Description:

- Enables the WWDG early wakeup interrupt.

Parameters:

- `__HANDLE__`: WWDG handle
- `__INTERRUPT__`: specifies the interrupt to enable. This parameter can be one of the following values:
 - `WWDG_IT_EWI`: Early wakeup interrupt

Return value:

- None

Notes:

- Once enabled this interrupt cannot be disabled except by a system reset.

Description:

- Disables the WWDG early wakeup interrupt.

Parameters:

- `__HANDLE__`: WWDG handle
- `__INTERRUPT__`: specifies the interrupt to disable. This parameter can be one of the following values:
 - `WWDG_IT_EWI`: Early wakeup interrupt

Return value:

- None

Notes:

- **WARNING:** This is a dummy macro for HAL code alignment. Once enabled this interrupt cannot be disabled except by a system reset.

Description:

- Gets the selected WWDG's it status.

Parameters:

- `__HANDLE__`: WWDG handle
- `__INTERRUPT__`: specifies the it to check. This parameter can be one of the following values:
 - `WWDG_FLAG_EWIF`: Early wakeup interrupt IT

Return value:

- The: new state of `WWDG_FLAG` (SET or RESET).

Description:

- Clear the WWDG's interrupt pending bits bits to clear the selected interrupt pending bits.

`__HAL_WWDG_DISABLE_IT`

`__HAL_WWDG_GET_IT`

`__HAL_WWDG_CLEAR_IT`

`__HAL_WWDG_GET_FLAG`**Parameters:**

- `__HANDLE__`: WWDG handle
- `__INTERRUPT__`: specifies the interrupt pending bit to clear. This parameter can be one of the following values:
 - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

Description:

- Gets the selected WWDG's flag status.

Parameters:

- `__HANDLE__`: WWDG handle
- `__FLAG__`: specifies the flag to check. This parameter can be one of the following values:
 - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

Return value:

- The: new state of `WWDG_FLAG` (SET or RESET).

Description:

- Clears the WWDG's pending flags.

Parameters:

- `__HANDLE__`: WWDG handle
- `__FLAG__`: specifies the flag to clear. This parameter can be one of the following values:
 - `WWDG_FLAG_EWIF`: Early wakeup interrupt flag

Return value:

- None

`__HAL_WWDG_CLEAR_FLAG`**Description:**

- Checks if the specified WWDG interrupt source is enabled or disabled.

Parameters:

- `__HANDLE__`: WWDG Handle.
- `__INTERRUPT__`: specifies the WWDG interrupt source to check. This parameter can be one of the following values:
 - `WWDG_IT_EWI`: Early Wakeup Interrupt

Return value:

- state: of `__INTERRUPT__` (TRUE or

`__HAL_WWDG_GET_IT_SOURCE`

FALSE).

WWDG Flag definition

WWDG_FLAG_EWIF Early wakeup interrupt flag

WWDG Interrupt definition

WWDG_IT_EWI Early wakeup interrupt

WWDG Prescaler

WWDG_PRESCALER_1 WWDG counter clock = (PCLK1/4096)/1

WWDG_PRESCALER_2 WWDG counter clock = (PCLK1/4096)/2

WWDG_PRESCALER_4 WWDG counter clock = (PCLK1/4096)/4

WWDG_PRESCALER_8 WWDG counter clock = (PCLK1/4096)/8

IS_WWDG_PRESCALER

WWDG Window

IS_WWDG_WINDOW

46 FAQs

General subjects

Why should I use the HAL drivers?

There are many advantages in using the HAL drivers:

- Ease of use: you can use the HAL drivers to configure and control any peripheral embedded within your STM32 MCU without prior in-depth knowledge of the product.
- HAL drivers provide intuitive and ready-to-use APIs to configure the peripherals and support polling, interrupt and DMA programming model to accommodate all application requirements, thus allowing the end-user to build a complete application by calling a few APIs.
- Higher level of abstraction than a standard peripheral library allowing to transparently manage:
 - Data transfers and processing using blocking mode (polling) or non-blocking mode (interrupt or DMA)
 - Error management through peripheral error detection and timeout mechanism.
- Generic architecture speeding up initialization and porting, thus allowing customers to focus on innovation.
- Generic set of APIs with full compatibility across the STM32 series/lines, to ease the porting task between STM32 MCUs.
- The APIs provided within the HAL drivers are feature-oriented and do not required in-depth knowledge of peripheral operation.
- The APIs provided are modular. They include initialization, IO operation and control functions. The end-user has to call init function, then start the process by calling one IO operation functions (write, read, transmit, receive, ...). Most of the peripherals have the same architecture.
- The number of functions required to build a complete and useful application is very reduced. As an example, to build a UART communication process, the user only has to call HAL_UART_Init() then HAL_UART_Transmit() or HAL_UART_Receive().

Which STM32L1 devices are supported by the HAL drivers?

The HAL drivers are developed to support all STM32L1 devices. To ensure compatibility between all devices and portability with others series and lines, the API is split into the generic and the extension APIs . For more details, please refer to [Section 4.4: "Devices supported by HAL drivers"](#).

What is the cost of using HAL drivers in term of code size and performance?

Like generic architecture drivers, the HAL drivers may induce firmware overhead.

This is due to the high abstraction level and ready-to-use APIs which allow data transfers, errors management and offloads the user application from implementation details.

Architecture

How many files should I modify to configure the HAL drivers?

Only one file needs to be modified: stm32l1xx_hal_conf.h. You can modify this file by disabling unused modules, or adjusting some parameters (i.e. HSE value, System configuration...)

A template is provided in the HAL drivers folders (stm32l1xx_hal_conf_template.c).

Which header files should I include in my application to use the HAL drivers?

Only stm32l1xx_hal.h file has to be included.

What is the difference between stm32l1xx_hal_ppp.c/h and stm32l1xx_hal_ppp_ex.c/h?

The HAL driver architecture supports common features across STM32 series/lines. To support specific features, the drivers are split into two groups.

- The generic APIs (stm32l1xx_hal_ppp.c): It includes the common set of APIs across all the STM32 product lines
- The extension APIs (stm32l1xx_hal_ppp_ex.c): It includes the specific APIs for specific device part number or family.

Initialization and I/O operation functions

How do I configure the system clock?

Unlike the standard library, the system clock configuration is not performed in CMSIS drivers file (system_stm32l1xx.c) but in the main user application by calling the two main functions, HAL_RCC_OscConfig() and HAL_RCC_ClockConfig(). It can be modified in any user application section.

What is the purpose of the *PPP_HandleTypeDef* **pHandle* structure located in each driver in addition to the Initialization structure

*PPP_HandleTypeDef *pHandle* is the main structure implemented in the HAL drivers. It handles the peripheral configuration and registers, and embeds all the structures and variables required to follow the peripheral device flow (pointer to buffer, Error code, State,...)

However, this structure is not required to service peripherals such as GPIO, SYSTICK, PWR, and RCC.

What is the purpose of HAL_PPP_MspInit() and HAL_PPP_MspDeInit() functions?

These function are called within HAL_PPP_Init() and HAL_PPP_DeInit(), respectively. They are used to perform the low level Initialization/de-initialization related to the additional hardware resources (RCC, GPIO, NVIC and DMA).

These functions are declared in stm32l1xx_hal_msp.c. A template is provided in the HAL driver folders (stm32l1xx_hal_msp_template.c).

When and how should I use callbacks functions (functions declared with the attribute *__weak*)?

Use callback functions for the I/O operations used in DMA or interrupt mode. The PPP process complete callbacks are called to inform the user about process completion in real-time event mode (interrupts).

The Errors callbacks are called when a processing error occurs in DMA or interrupt mode. These callbacks are customized by the user to add user proprietary code. They can be declared in the application. Note that the same process completion callbacks are used for DMA and interrupt mode.

Is it mandatory to use HAL_Init() function at the beginning of the user application?

It is mandatory to use HAL_Init() function to enable the system configuration (Prefetch, Data instruction cache,...), configure the SysTick and the NVIC priority grouping and the hardware low level initialization.

The SysTick configuration shall be adjusted by calling **HAL_RCC_ClockConfig()** function, to obtain 1 ms whatever the system clock.

Why do I need to configure the SysTick timer to use the HAL drivers?

The SysTick timer is configured to be used to generate variable increments by calling **HAL_IncTick()** function in SysTick ISR and retrieve the value of this variable by calling **HAL_GetTick()** function.

The call HAL_GetTick() function is mandatory when using HAL drivers with Polling Process or when using HAL_Delay().

Why is the SysTick timer configured to have 1 ms?

This is mandatory to ensure correct IO operation in particular for polling mode operation where the 1 ms is required as timebase.

Could HAL_Delay() function block my application under certain conditions?

Care must be taken when using HAL_Delay() since this function provides accurate delay based on a variable incremented in SysTick ISR. This implies that if HAL_Delay() is called from a peripheral ISR process, then the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt, otherwise the caller ISR process will be blocked. Use HAL_NVIC_SetPriority() function to change the SysTick interrupt priority.

What programming model sequence should I follow to use HAL drivers ?

Follow the sequence below to use the APIs provided in the HAL drivers:

1. Call HAL_Init() function to initialize the system (data cache, NVIC priority,...).
2. Initialize the system clock by calling HAL_RCC_OscConfig() followed by HAL_RCC_ClockConfig().
3. Add HAL_IncTick() function under SysTick_Handler() ISR function to enable polling process when using HAL_Delay() function
4. Start initializing your peripheral by calling HAL_PPP_Init().
5. Implement the hardware low level initialization (Peripheral clock, GPIO, DMA,...) by calling HAL_PPP_MspInit() in stm32l1xx_hal_msp.c
6. Start your process operation by calling IO operation functions.

What is the purpose of HAL_PPP_IRQHandler() function and when should I use it?

HAL_PPP_IRQHandler() is used to handle interrupt process. It is called under PPP_IRQHandler() function in stm32l1xx_it.c. In this case, the end-user has to implement only the callbacks functions (prefixed by __weak) to perform the appropriate action when an interrupt is detected. Advanced users can implement their own code in PPP_IRQHandler() without calling HAL_PPP_IRQHandler().

Can I use directly the macros defined in stm32l1xx_hal_ppp.h ?

Yes, you can: a set of macros is provided with the APIs. They allow accessing directly some specific features using peripheral flags.

Where must PPP_HandleTypedef structure peripheral handler be declared?

PPP_HandleTypedef structure peripheral handler must be declared as a global variable, so that all the structure fields are set to 0 by default. In this way, the peripheral handler default state are set to HAL_PPP_STATE_RESET, which is the default state for each peripheral after a system reset.

47 Revision history

Table 26: Document revision history

Date	Revision	Changes
18-Nov-2014	1	Initial release.
13-Feb-2015	2	Updated list of PPP_TypeDef structures in Section 4.5.1: "HAL API naming rules" . Updated Table 5: "List of devices supported by HAL drivers" . Updated Section 19.3.4: "HAL_FLASHEx_Erase" , Section 19.3.5: "HAL_FLASHEx_Erase_IT" and Section 19.2.2: "FLASH_OBProgramInitTypeDef" .

Date	Revision	Changes
27-May-2015	3	<p>Changed path for compilation under Unix environment.</p> <p>Updated common macros in Section 2:9 "Common resources".</p> <p>Updated drivers to be C++ compliant.</p> <p>Added interface to access MPU features (refer to stm32l1xx_hal_cortex.h).</p> <p>Section 12: "HAL CRYPT Generic Driver": added instance field in CRYPT_HandleTypeDef.</p> <p>Section 18: "HAL FLASH Generic Driver": changed field name of NOR_CFITypeDef (CFI1X changed to CFI1_X).</p> <p>Section 31: "HAL PCD Generic Driver":</p> <ul style="list-style-type: none"> HAL_PCD_ActiveRemoteWakeup renamed HAL_PCD_ActivateRemoteWakeup HAL_PCD_DeActiveRemoteWakeup renamed to HAL_PCD_DeActivateRemoteWakeup HAL_PCD_ActiveRemoteWakeup renamed HAL_PCD_ActivateRemoteWakeup HAL_PCD_DeActiveRemoteWakeup renamed to HAL_PCD_DeActivateRemoteWakeup. <p>Section 33: "HAL PWR Generic Driver":</p> <ul style="list-style-type: none"> HAL_PWR_PVDConfig renamed HAL_PWR_ConfigPVD. Added new interfaces: <ul style="list-style-type: none"> void HAL_PWR_EnableSleepOnExit(void); void HAL_PWR_DisableSleepOnExit(void); void HAL_PWR_EnableSEVOnPend(void); void HAL_PWR_DisableSEVOnPend(void); void HAL_PWR_EnableSleepOnExit(void); uint32_t HAL_PWREx_GetVoltageRange(void); <p>Section 35: "HAL RCC Generic Driver":</p> <ul style="list-style-type: none"> HAL_RCC_CCSCallback renamed to HAL_RCC_CSSCallback. Added HAL_RCCEX_GetPeriphCLKFreq interface. <p>Section 40: "HAL SMARTCARD Generic Driver": Removed HAL_SMARTCARD_ReInit interface.</p> <p>Section 41: "HAL SPI Generic Driver": HAL_SPI_GetError now returns a uint32_t instead of HAL_SPI_ErrorTypeDef.</p> <p>Section 43: "HAL TIM Generic Driver": added HAL_TIM_SlaveConfigSynchronization_IT interface.</p> <p>Section 45: "HAL UART Generic Driver": Changed ErrorCode field of UART_HandleTypeDef from HAL_UART_ErrorTypeDef to uint32_t.</p> <p>Section 46: "HAL USART Generic Driver": Changed ErrorCode field of UART_HandleTypeDef from HAL_UART_ErrorTypeDef to uint32_t.</p>

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved